Faculdade de Engenharia da Universidade do Porto

Estratégias Híbridas de *Machine Learning* e Simulação para a Resolução de Problemas de Escalonamento

Romão Filipe Dias Santos

VERSÃO FINAL

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores
Major Automação

Orientador: Jorge Manuel Pinho De Sousa
Coorientador: Samuel De Oliveira Moniz

25 de Junho de 2018

Resumo

Esta dissertação propõe uma abordagem híbrida de *Machine Learning* e simulação, para a resolução de problemas de escalonamento da produção, considerando tempos de processamento determinísticos e *buffers* de capacidade limitada. A abordagem seguida combina modelos de simulação com técnicas de *Reinforcement Learning*. De entre estas técnicas, destaca-se a *Neuroevolution* que combina Redes Neuronais com Algoritmos Genéticos. Para modelar os problemas reais de escalonamento da produção, considerou-se fundamental o uso de simulação de eventos discretos.

Os resultados computacionais obtidos mostram que a conjugação dos modelos de simulação e de *Neuroevolution* permite obter níveis de desempenho superiores, quando em comparação com as regras de prioridade (*dispatching rules*) da literatura. São ainda retiradas conclusões quanto à eficiência do método desenvolvido e à sua utilidade em contexto real.

Abstract

This dissertation proposes a hybrid approach of Machine Learning and Simulation, to solve production scheduling problems, considering deterministic processing times and limited capacity buffers. The approach followed combines simulation models with Reinforcement Learning techniques. Among these techniques, Neuroevolution combines Neural Networks with Genetic Algorithms. In order to model the real problems of production scheduling, the use of discrete event simulation was considered fundamental.

The computational results show that the combination of the simulation and Neuroevolution models allows to obtain higher performance levels when compared to the dispatching rules of the literature. Further conclusions are drawn regarding the efficiency of the method developed and its usefulness in real context.

Agradecimentos

Como qualquer coisa que se deseje na vida, para que exista sucesso é preciso esforço, dedicação e esperança. Foi com esta mentalidade que foi realizada esta dissertação, tendo sempre em consideração todos aqueles que contribuíram direta ou indiretamente para o seu desenvolvimento.

Quero deixar desde já o meu muito obrigado ao meu orientador, o Professor Doutor Jorge Manuel Pinho De Sousa e ao meu coorientador, o Professor Doutor Samuel De Oliveira Moniz, pelo fantástico apoio recebido ao longo destes longos meses. O caráter fantástico de ambos, combinado com um profissionalismo exemplar, tornaram a realização da minha dissertação possível. Não tenho dúvidas que todos os tempinhos que sempre tiveram para me ajudar foram cruciais para que a minha dissertação levasse o rumo certo.

Também quero deixar o meu agradecimento ao INESC TEC – Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência – pelas instalações cedidas e quero também agradecer ao pessoal do Centro de Engenharia de Sistemas Empresariais (CESE) pelo fantástico ambiente de boa disposição que me proporcionaram. Aproveito também para agradecer em especial ao Rúben Dias, pelo tempo que despendeu para me ajudar e pelos ensinamentos passados. Também agradeço ao Tiago Santos, uma vez que contribuiu com um modelo para a presente dissertação e por ser um dos meus melhores amigos, com quem passei muitos e bons momentos. Deixo também um agradecimento especial ao meu grande amigo João Basto, pela ajuda na revisão da tese aos finos e bons momentos na AE da FEUP.

Aproveito também a oportunidade de agradecer às pessoas mais importantes da minha vida. Aos meus pais que tanto se sacrificaram e trabalharam para que eu tivesse um futuro risonho, muito obrigado e espero nunca vos desapontar. À minha namorada, por todas as vezes que a dissertação interferiu nas nossas coisas e que sempre soube me apoiar e dar a força para ultrapassar todos os obstáculos. Muito obrigado pela pessoa que és e espero que continues do meu lado, porque só assim é que sou forte. Ao meu irmão, por ser o meu companheiro de ginásio e meu amigo, ao qual tenho muito orgulho e me apoia muito.

Como não poderia deixar de ser, quero agradecer a todos os meus amigos e familiares, não particularizando, pois este documento não permite um agradecimento completo e todos eles sabem a importância que têm. Um muito obrigado sincero a todos!

Romão Santos

"We cannot solve our problems with the same thinking we use when we create them"

Albert Einstein

Índice

Lista de figuras

Lista de tabelas

Abreviaturas e Símbolos

AGV	*Automatic Guided vehicle*
API	*Application Programming Interface*
NE	*Neuroevolution*
RL	*Reinforcement Learning*

Capítulo 1

Introdução

Este capítulo introdutório tem como objetivo a apresentação do tema de dissertação, "Estratégias Híbridas de *Machine Learning* e Simulação para a Resolução de Problemas de Escalonamento". Na secção 1.1 é feita a sua contextualização, sendo os principais objetivos a alcançar aquando da sua realização abordados na secção posterior 1.2. Avançando para a secção 1.3 são salientadas as motivações que suportaram a proposição do tema, enquanto na seção 1.4 é apresentado um sumário da metodologia traçada. Para finalizar o capítulo é apresentada a estruturação do presente documento na secção 1.5, onde são destacados os principais focos dos capítulos sequentes.

1.1 - Contextualização

Para que uma organização tenha sucesso é essencial que conquiste uma mentalidade assente na melhoria contínua e com estratégia competitiva face à concorrência. Para enfrentar o mercado as empresas investem recorrentemente em métodos de controlo, planeamento e escalonamento. Neste contexto, as diferentes empresas ligadas à indústria tendem cada vez mais a aprimorar os seus métodos de otimização da produção.

Focando na melhoria da eficiência de um sistema de produção e nas vantagens associadas, como a redução de custos ou o aumento da produtividade, torna-se fundamental por parte das empresas o estudo de métodos inovadores.

O escalonamento da produção, visto como uma ferramenta importante em sistemas de manufatura, consegue causar impacto na produtividade e no sistema de produção [1]. Pinedo [2] refere o escalonamento de produção como sendo um processo de *decision-making* usado por inúmeras indústrias de serviços e manufatura. O escalonamento lida com a alocação de recursos a tarefas, tendo em conta um dado período de tempo e o fundamento baseia-se em otimizar um ou mais objetivos.

No processo de escalonamento é fundamental saber a quantidade e o tipo de recursos disponíveis para que um conjunto de tarefas sejam cumpridas. Também é essencial que se definam as tarefas que serão executadas em cada recurso, a sua duração, o ponto temporal de partida e o período onde efetivamente deve estar concluída. A informação sobre os recursos e

tarefas definem maioritariamente um problema de escalonamento, ainda assim é relevante considerar que encontrar uma solução ótima assume-se com sendo um processo complexo derivado de diferentes complexidades do sistema de produção [3].

O uso de aplicações de *Machine Learning* em sistemas de produção é uma tendência relativamente recente, no entanto, cada vez mais são os casos de sucesso de aplicações de *Machine Learning* em processos de otimização, controlo e previsão de falhas. Estes métodos revelam-se interessantes especialmente pela capacidade de resolução de problemas complexos. Quando aplicadas com sucesso, as técnicas de *Machine Learning* são capazes de lidar com uma grande dimensão e variedade de dados e retirar relações implícitas em ambientes complexos [4].

Face às vantagens associadas aos métodos de *Machine Learning* para otimização do processo de produção de uma empresa, mais concretamente o escalonamento da produção no qual se insere esta dissertação, assume-se viável explorar alternativas deste tipo para resolução de eventuais problemas de escalonamento.

1.2 - Objetivos

O principal objetivo desta dissertação passa pela elaboração de um método inovador, baseado em *Machine Learning* e simulação, para resolução de problemas de escalonamento de um chão de fábrica.

Tendo em conta um determinado número de peças a serem produzidas num específico período de tempo, é necessário escalonar as ordens de produção face aos recursos disponíveis. No sentido de avaliar a eficácia da abordagem, será considerado como objetivo incrementar a produtividade, isto é, o número de peças total ao fim de um determinado tempo.

O uso da simulação de eventos discretos tem como fundamento não só avaliar comportamentos num contexto de chão de fábrica, mas também compreender e testar diferentes cenários de cada caso de estudo. A simulação será implementada no programa SIMIO.

O objetivo de resolver problemas de escalonamento usando técnicas de *Machine Learning* passa pela tentativa de obter soluções viáveis ao fim de um determinado tempo. É esperado que o conjunto de soluções façam sentido e sejam aplicáveis nos casos de estudo.

O recurso à simulação integrada com técnicas de *Machine Learning* tem como fundamento avaliar a possibilidade de recorrer a este tipo de abordagens em casos reais e será esperado obter resposta sobre a eficiência da combinação das duas técnicas.

Por fim, para avaliar o desempenho da metodologia, será feita uma comparação com regras de prioridade que funcionam como referência de decisões simples, normalmente consideradas em contexto real.

1.3 - Motivação

Ao longo do tempo soluções inovadoras e revolucionárias na área dos processos industriais foram sendo criadas, desde a abordagem *Lean* desenvolvida por Taiichi Ohno, baseada na eliminação de desperdício e criação de valor, até à estratégia *Kaizen* desenvolvida por Masaaki Imai, assente na melhoria contínua e no trabalho de equipa em sistemas de produção.

Para o sucesso de qualquer indústria é inevitável a otimização dos processos de produção e o uso de técnicas inovadoras que as diferenciem no mercado. Como já referido, o

escalonamento da produção é crucial para que uma empresa seja competitiva e obtenha resultados superiores à concorrência. Um escalonamento de produção traduz-se como sendo a alocação eficiente de várias tarefas a desempenhar, tendo em conta os recursos disponíveis, num determinado período de tempo.

Focando nas vantagens associadas à melhoria dos sistemas de produção e no sucesso de medidas para otimizar a produção, a aposta em técnicas promissoras baseadas em *Machine Learning* são cada vez usadas mais frequentemente a pensar no futuro da indústria.

Sem ser necessário um investimento em mais recursos, materiais ou trabalhadores, é possível melhorar a eficiência de um sistema de produção apenas focando no desenvolvimento de ferramentas que mudem o paradigma de uma empresa e mitiguem os argumentos dos reticentes à mudança.

É interessante salientar os baixos custos associados à construção de escalonamentos de produção, visto que com o mesmo número de recursos é possível obter ganhos superiores e manter ou aumentar a qualidade que o cliente espera.

As técnicas tradicionais usadas para escalonamento apresentam algumas limitações quando aplicadas em problemas industriais, onde os mixes de produção apresentam um crescimento na diversidade de produtos e uma diminuição nos lotes para cada produto. Isto, aliado ao facto de os sistemas terem de ser altamente responsivos a novos dados (como por exemplo novas encomendas) que estão em constante alteração, leva a crer que é necessário construir uma abordagem de escalonamento mais capaz de se adaptar e responder às incertezas nas circunstâncias do processo produtivo. Assim, ao contrário de heurísticas e modelos de otimização, que funcionam sempre *a priori* e que utilizando os dados disponíveis fazem o escalonamento para um horizonte futuro, a utilização de redes neuronais de forma responsiva, que utilizam a cada momento as características correntes do sistema para tomar a decisão, pode ser uma boa resposta para as novas exigências do mercado da produção industrial.

É esperado que a combinação de simulação de contextos reais combinadas com técnicas inovadoras ajude a indústria a melhorar os seus processos e a evoluir, no sentido de alcançar novas metas e melhores resultados, estabelecendo assim a motivação deste trabalho.

1.4 - Metodologia

Sendo este tópico do documento apenas para introduzir a metodologia a seguir, é importante destacar que serve de sumário ao que vai ser explicado com mais detalhe posteriormente.

A construção da metodologia surgiu com base nos objetivos a atingir no ponto 1.2 deste capítulo e tem como intuito tentar resolver problemas de escalonamento da produção.

A metodologia seguida combina modelos de Simulação com técnicas de *Reinforcement Learning*. De entre estas técnicas, destaca-se a *Neuroevolution* que usa algoritmos evolucionários, neste caso Algoritmos Genéticos, para construir e melhorar Redes Neuronais. Para modelar os problemas reais de escalonamento da produção, considerou-se fundamental o uso de simulação de eventos discretos.

A integração de *Neuroevolution* com simulação de eventos discretos surge a partir de funcionalidades do programa escolhido, Simio, que permite a incorporação de código externo para controlo, planeamento e customização de etapas, propriedades ou objetos.

1.5 – Estrutura do documento

A divisão do presente documento foi baseada no trabalho produzido ao longo do projeto, tendo-se traduzido num total de seis capítulos. O presente capítulo serve de introdução ao documento e foca-se em contextualizar, estruturar e determinar os objetivos a atingir.

No capítulo 2, referente à revisão bibliográfica, é apresentada uma pesquisa detalhada sobre o escalonamento de um chão de fábrica e de métodos de *Machine Learning* aplicados a escalonamento, dando especial atenção a métodos de resolução de problemas *Reinforcement Learning*. Também será feita uma revisão sobre os efeitos do uso de simulação para otimização de produção e produzida uma análise conclusiva acerca das melhores alternativas a ter em conta.

O capítulo 3 refere-se à abordagem metodológica e visa compreender de uma forma genérica qual a abordagem a ser implementada no projeto, tendo em conta a pesquisa bibliográfica realizada.

No capítulo 4, serão estruturados os casos de estudo em que vai ser aplicada a metodologia desenvolvida.

Com a definição dos casos de estudo e implementação da abordagem metodológica, surge a análise de resultados em cada um dos casos de estudo no capítulo 5.

Por último, o capítulo 6 remete à conclusão e trabalho futuro, como forma de análise ao trabalho realizado no projeto e de perspetivas para eventuais iniciativas no seguimento do projeto.

Capítulo 2

Revisão bibliográfica

Este capítulo referente à revisão bibliográfica é consequência da pesquisa realizada, com intuito de compreender grande parte dos temas constituintes da dissertação. Tendo em conta os objetivos a atingir e considerando todos os parâmetros do contexto e motivação, foi o capítulo subdividido por temas.

Primeiramente no ponto 2.1 será descrita e analisada a importância do escalonamento ao nível de um chão de fábrica, assim como os métodos normalmente usados. Sendo que cada vez mais a inteligência computacional desempenha um papel satisfatório e inovador ao nível do escalonamento, surge o interesse de estudo de métodos de *Machine Learning* no ponto 2.2.

Visto não estar definido um conjunto de dados rotulados, sendo o objetivo da aprendizagem obter a maior recompensa possível é notório que se trata de um problema do tipo *Reinforcement Learning*, ao qual irá ser desenvolvido o seu conceito, bem como comparação com outros tipos de aprendizagem como *Supervised Learning* e *Unsupervised Learning*.

No ponto 2.3 são detalhados diferentes métodos para resolução de problemas do tipo *Reinforcement Learning*, destacando-se o *Q-learning* e o *Neuroevolution*, sendo este último base para a abordagem seguida.

Por fim é exibido no ponto 2.4 a pesquisa relativa à simulação de um chão de fábrica, tendo como fundamento a sua importância e a viabilidade do uso do programa Simio para fins de simulação.

2.1 – Escalonamento da produção

Brucker [5] realça que o escalonamento de um sistema de fabrico está dependente de ordens de fabrico, da maquinaria de trabalho, dos recursos disponíveis e das operações a realizar em cada máquina, sendo que em cada máquina poderão haver tempos de processamento distintos. O objetivo é planear e alocar eficientemente a maquinaria e os recursos disponíveis às operações a executar.

2.1.1 - Problemas de escalonamento

Supondo que *m* máquinas $M_j(j = 1, \dots, m)$ necessitam executar *n* ordens de produção $J_i(i = 1, \dots, n)$, um cronograma é definido para cada ordem de produção com uma alocação de um ou mais intervalos de tempo a uma ou mais máquinas. Para representação de cronogramas podem ser usados diagramas de Gantt, podendo ser orientados às máquinas como está visível

na figura 2.1 a) ou às ordens de produção, figura 2.1 b). O problema de escalonamento consiste em encontrar um cronograma que satisfaça determinadas restrições.

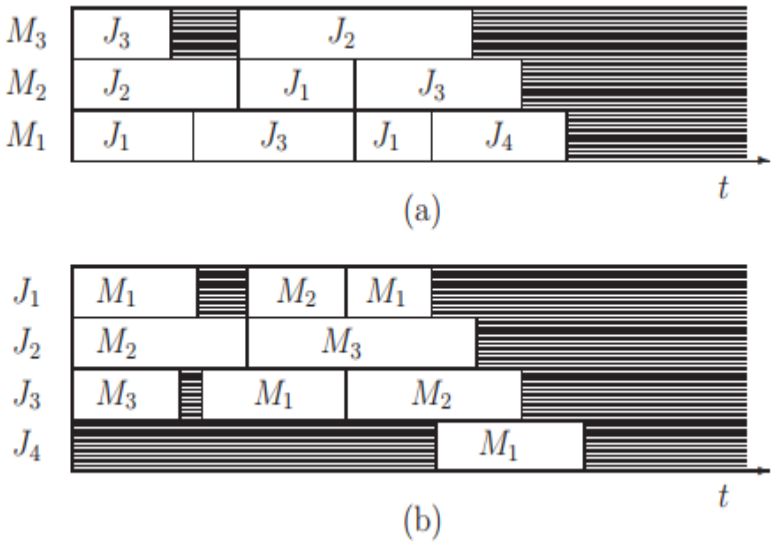

Figura 2.1 - Diagramas de Gantt orientado a máquinas e a ordens de produção, adaptado de [5]

2.1.2 – Ordens de produção

Uma ordem de produção consiste num número n_i de operações $O_{i1}, \dots, O_{i,ni}$. A cada operação O_{ij} está associada necessariamente um processamento p_{ij}. Caso uma ordem de operação J_i apenas seja composta por uma operação $(n_i = 1)$, é identificado J_i com O_{i1} e o respetivo processamento p_i. Além disto, também uma data de lançamento r_i pode ser especificada, onde a primeira operação de J_i se encontra disponível para processamento. Associado a cada operação O_{ij} está um conjunto de máquinas $\mu_{ij} \subseteq \{M_1, \dots, M_m\}$. O_{ij} pode ser processada em qualquer das máquinas do conjunto μ_{ij}. Normalmente, todos os μ_{ij} são conjuntos de um elemento ou todos os μ_{ij} são iguais ao conjuntos de todas as máquinas. Para o primeiro conjunto as máquinas são apelidadas de específicas, enquanto para o segundo caso são tratadas como máquinas em paralelo.

Por fim, é definida uma função de custo $f_i(t)$ que mede o peso de completar uma ordem de produção J_i num tempo t. Um escalonamento é viável se não existirem dois intervalos de tempo sobrepostos na mesma máquina, se não existirem dois intervalos de tempo sobrepostos para a mesma ordem de produção e adicionalmente se cumprir um número de características específicas do problema. Um escalonamento é ótimo se minimizar um critério previamente estabelecido.

2.1.3 – Classificação de problemas de escalonamento de fábrica

Um problema geral de modelação de processos de produção industriais é definido por *n* ordens de produção $i = 1, \dots, n$ e *m* máquinas $M_1, \dots, M_m$. Cada ordem de produção *i* consiste num conjunto de operações O_{ij} $(j = 1, \dots, n_i)$ com tempos de processamento p_{ij}. Cada operação O_{ij} deverá ser processada numa máquina $\mu_{ij} \in \{M_1, \dots, M_m\}$. Poderão existir precedências entre as operações de todas as ordens de produção. Cada ordem de produção apenas pode ser

processada por uma máquina de cada vez e cada máquina apenas pode processar uma ordem de cada vez. O objetivo é encontrar um escalonamento viável que minimize uma dada função objetivo composta pelos tempos de conclusão C_i de cada ordem de produção $i = 1, \dots, n$.

Os problemas de modelação de processos de produção industriais gerais podem ser distintos e classificados como problemas *open shop, flow shop ou job shop.*

Problemas *open shop*

Este tipo de problemas, derivado do problema geral de escalonamento, tem como principais caraterísticas:

- Cada ordem de produção $i = 1, \dots, n$ é composta por *m* operações O_{ij} $(j = 1, \dots, n_i)$, onde O_{ij} deve ser processada numa máquina M_j;
- Não existe relações de precedência entre as operações.

O problema consiste em determinar um sequenciamento de ordens de produção (sequenciamento de operações que pertence á mesma ordem de produção) e sequenciamento de máquinas (sequenciamento de operações a serem processadas na mesma máquina).

Problemas *flow shop*

No que diz respeito a problemas deste tipo, realçam-se um conjunto de caraterísticas:

- Cada ordem de produção $i = 1, \dots, n$ é composta por *m* operações O_{ij} com tempos de processamento p_{ij} $(j = 1, \dots, n_i)$, onde O_{ij} deve ser processada numa máquina M_j;
- Existe condições de precedência do tipo $O_{ij} \rightarrow O_{i,j+1}$ $(i = 1, \dots, m-1)$ para cada $i = 1, \dots, n$, isto é, cada ordem de produção é processada primeiro na máquina 1, depois na máquina 2, depois máquina 3, etc.

Este tipo de problema consiste em encontrar um sequenciamento de ordens de produção π_j para cada máquina ***j***.

Problemas *job shop*

Os problemas deste tipo são um caso especial dos problemas gerais de modelação de processos de produção industriais em que:

- Existem *n* ordens de produção $i = 1, \dots, n$ e *m* máquinas $M_1, \dots, M_m$;
- Cada ordem de produção $i = 1, \dots, n$ é composta por uma sequência de n_i operações $O_{i1}, O_{i2}, \dots, O_{in_i}$e existe condições de precedência do tipo $O_{ij} \rightarrow O_{i,j+1}$ $(j = 1, \dots, n_i - 1)$;
- Existe uma máquina $\mu_{ij} \in \{M_1, \dots, M_m\}$ e um tempo de processamento p_{ij} associado a cada operação O_{ij}, sendo que a operação O_{ij} deve ser processada por p_{ij} unidades de tempo na máquina μ_{ij};

O problema reside em encontrar um escalonamento viável que minimize uma função objetivo, dependendo dos tempos de conclusão C_i das últimas operações O_{i,n_i} das ordens de produção.

Tendo por base as caraterísticas previamente descritas de cada tipo de problema e tendo como referência o tipo de problema descrito neste documento, surgiu a necessidade de um

estudo aprofundado dos problemas de escalonamento *job shop* assim como os métodos usados para a sua resolução.

2.1.3 – Problemas de escalonamento do tipo *job shop*

O problema de escalonamento *job shop* é um dos mais conhecidos dos problemas de escalonamento. O problema é descrito por um conjunto de *n* ordens de produção $\{J_1, \dots, J_n\}$ a serem processadas num conjunto de *m* máquinas. Cada ordem carece de que um conjunto de operações consecutivas sejam terminadas (roteamento). O objetivo é determinar um cronograma, isto é, um tempo de conclusão C_j para cada ordem de produção J_j que satisfaça as restrições e minimize um dado critério. Um dos critérios mais tratado frequentemente é o *makespan* $(J||C_{max})$, isto é, o máximo dos tempos de conclusão para todas as tarefas ($C_{max} = max_j C_j$) [6].

Existe habitualmente um conjunto de premissas comuns no que diz respeito a problemas de escalonamento *job shop*:

- No início todas as ordens de produção estão prontas para começar;
- Os tempos de processamento são fixos e incluem os tempos de configuração;
- Uma máquina apenas pode processar uma tarefa de cada vez, e uma tarefa não pode ser realizada em mais que uma máquina ao mesmo tempo;
- Uma operação consiste na tarefa a desempenhar na máquina e existe restrições de precedência entre as operações, isto é, uma operação não pode começar sem que a outra tenha terminado;
- Todas as operações devem ser continuadas até que o tempo de processamento seja cumprido;
- As máquinas estão sempre disponíveis, não havendo avarias ou manutenção das mesmas.

O problema de escalonamento *job shop* é tratado como um problema sequencial, tendo como objetivo determinar a ordem das operações nas máquinas [7].

Para resolução de problemas de escalonamento existem distintas abordagens, podendo se optar por categorias tradicionais, ou recorrendo a outro tipo de métodos como é o caso de uma abordagem de simulação ou recorrendo ao uso de inteligência artificial [8].

Na figura 2.2 são apresentados um conjunto de métodos tradicionais usados na resolução de problemas de escalonamento do tipo *job shop*. De seguida irão ser abordados alguns deles bem como as limitações de cada um.

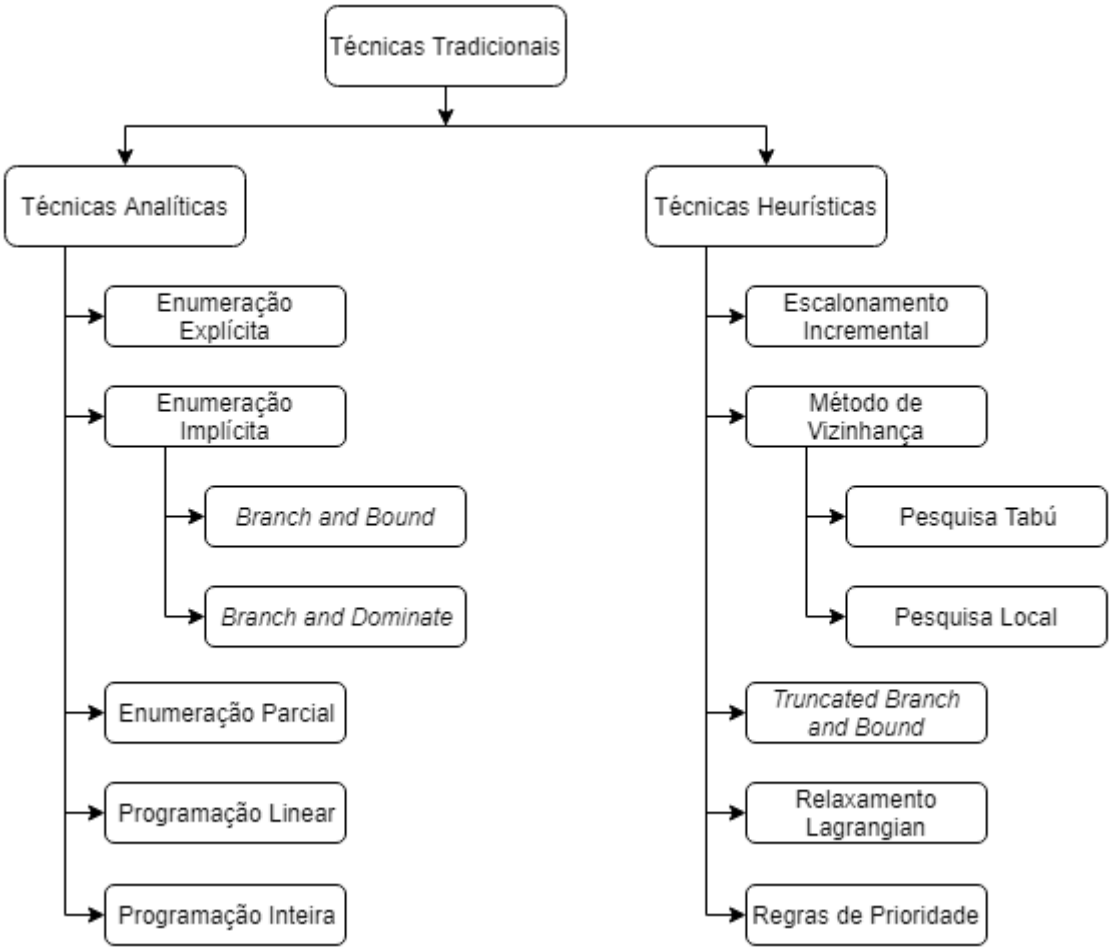

Figura 2.2 - Técnicas tradicionais para resolução de problemas de escalonamento *job shop*, adaptado de [8]

2.1.3.1 – Abordagem Analítica

A abordagem analítica interpreta o problema de escalonamento como um modelo de otimização, tendo em conta uma função objetivo e um conjunto de restrições explícitas. É elaborado um algoritmo apropriado para resolver o modelo [9].

Enumeração Explícita

Este método consiste em gerar uma árvore de enumeração completa baseada na pesquisa a realizar e no número de operações. As folhas das árvores representam todas as possíveis soluções. O caminho desde a raiz até à folha cujo *makespan* é mínimo representa uma solução ótima.

Limitação do método:

- A maior dificuldade reside no tamanho da árvore de pesquisa gerada. Uma vez que se tem um máximo de $(n!)^m$ soluções a considerar, mesmo problemas de tamanho moderado, manterão um computador ocupado durante um período de tempo que excede séculos [10].

Enumeração Implícita

A estratégia da enumeração implícita baseia-se em minimizar uma função objetivo sem considerar todas as possíveis soluções. Os esquemas de enumeração implícita analisam subgrupos cada vez mais pequenos de possíveis soluções, até que esses subgrupos não contenham soluções aprimoradas.

Os algoritmos *Branch and Bound* (B&B) cortam os ramos da árvore de enumeração e reduzem o número de nós gerados substancialmente. Algoritmos B&B dependem de um limite inferior e de um limite superior do valor da função objetivo. A melhor solução gerada determina o atual limite superior e um limite inferior é calculado para cada nó da árvore, começando pela raiz. A figura 2.3 representa um exemplo de uma árvore de enumeração com as decisões associadas e a respetiva profundidade. Os nós a cinzento representam uma decisão ao longo da árvore.

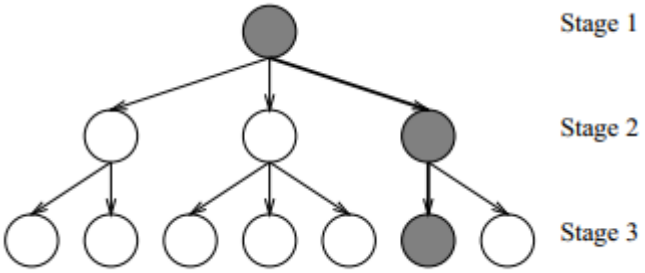

Figura 2.3 - Árvore de decisão com profundidade 3, adaptado de [10]

Limitações do método:

- Apenas o *makespan* pode ser avaliado pelo método;
- Existe uma dependência elevada da eficiência do limite inferior. Quanto mais eficiente o limite, menor o número de soluções da árvore que necessitam de ser executados;
- O método está limitado a um problema com algumas centenas de operações [10].

O método *Branch and Dominated* é similar ao B&B e apenas difere na abordagem de corte dos ramos. Se existir um conjunto de condições num nó que signifique que o escalonamento é inferior ao melhor caso de outro nó, o primeiro nó é eliminado para as restantes considerações, sendo que desta forma o segundo nó domina o primeiro. Este método está limitado pelas limitações computacionais derivadas da redução da pesquisa pelas condições de domínio. Tal como o B&B é impraticável enumerar um conjunto reduzido [11].

Enumeração Parcial

Um escalonamento ótimo encontra-se sempre presente num conjunto de escalonamentos viáveis, denominados “ativos”. Esta denominação de escalonamentos viáveis ativos foi usada por Grabowski [12]. O algoritmo adiciona iterativamente uma seleção de máquinas ao escalonamento parcial. Apesar do método usado reduzir o peso computacional, está ainda limitado pela necessidade de gerar um grande número de escalonamentos para encontrar um

que seja ótimo. A complexidade do problema cresce com a inserção de mais máquinas e mais ordens de produção.

Programação Linear

Este tipo de abordagem analítica é particularmente atrativa ao nível do modelo desenvolvido. Programas altamente eficientes conseguem resolver problemas com inúmeras variáveis e grandes conjuntos de restrições. Em alguns casos de problemas específicos, algumas aproximações simplificadas levam a uma solução de programação linear. Contudo a programação linear está limitada por alguns requisitos do método e apenas pode ser usada se o problema cumprir inteiramente essas limitações. O facto de um problema ser aproximado também é considerado uma limitação, sendo que um problema real de planeamento e escalonamento fabril não se comporta linearmente.

Programação Inteira

Na tentativa de resolver as limitações da programação linear, este método usa variáveis inteiras, necessitando de algoritmos menos eficientes. Wagner [13] tratou problemas de escalonamento *job shop* usando programação inteira.

2.1.3.2 – Abordagem heurística

A maior parte das limitações da abordagem analítica para resolução de problemas de *escalonamento job shop* está relacionada com o esforço computacional necessário. A abordagem heurística consegue obter boas soluções com menos esforço computacional. Sabendo que os métodos como a programação dinâmica ou o B&B são capazes de obter soluções mais robustas, a aplicação de técnicas heurísticas é relativamente mais simples e eficazes de implementar [3].

Escalonamento Incremental

Um escalonamento incremental começa com a construção de uma linha temporal vazia e um conjunto de tarefas a serem agendadas. A base deste método consiste em determinar a próxima tarefa a agendar e adicioná-la à linha temporal para que nenhuma limitação seja violada. Dependendo do grau de otimização, o algoritmo de posicionamento poderá ser simples ou elaborado. Este processo é repetido até que todas as tarefas sejam agendadas ou caso não exista tempo para agendar mais nenhuma tarefa.

Este método encontra por vezes pontos sem saída, em que são apresentadas soluções incompletas. Sendo esta uma limitação importante a solução encontrada é baseada em tentativa e erro, o que torna a otimização pobre e lenta.

Métodos de vizinhança

Um método de vizinhança começa por encontrar um escalonamento viável, ajustando-o continuamente para que os ajustes realizados o melhorem. Para ajustar ciclicamente o escalonamento, são procuradas soluções na vizinhança da solução viável. Podem-se distinguir duas bases importantes neste método: a sequência de vizinhança e a geração de mecanismos de vizinhança para essa sequência.

Alguns métodos dentro desta gama foram estudados, como é o caso da pesquisa tabu [14], Esta técnica foi aplicada em problemas de escalonamento *job shop,* tendo obtido resultados interessantes e mais eficientes que técnicas como o S*imulated Annealing*. No entanto este

método está restringido pela larga memória computacional necessária, derivado aos conjuntos de caminhos de soluções serem mantidos em memória [10].

Também outros métodos de pesquisa local são implementados na categoria de métodos de vizinhança, como é o caso de *Simulated Annealing* (SA). Este método, assim como a pesquisa tabu, constroem a vizinhança com base em arranjos no escalonamento. SA apresenta um consumo temporal muito superior à pesquisa tabu em certas circunstâncias [16].

Truncated Branch and Bound

Este método de aproximação é um dos métodos desenvolvidos mais eficiente para procedimentos *shifting bottleneck* [15]. A ideia baseia-se em começar por um problema de escalonamento *job shop* inicial, otimizar a sequência de máquinas uma a uma e aplicar o algoritmo de Carlier [16] aplicado a problemas de uma máquina. A ordem pela qual as máquinas são sequenciadas depende da medida do *bottleneck* associada a elas. Este procedimento faz parte de heurísticas do tipo enumerativo, em que cada nó de pesquisa representa um conjunto de sequência de máquinas. Comparando com outros algoritmos, este é menos eficiente pois cada vez que é sequenciada uma máquina, é necessário rever todo o escalonamento feito previamente, tornando o processo longo.

Relaxamento Lagrangiano

Metodologias baseadas em relaxamento lagrangiano foram implementadas em problemas de escalonamento *job shop* com máquinas em paralelo e foram conseguidos resultados quase ótimos sendo computacionalmente eficientes. Foi aplicado a escalonamentos com múltiplas máquinas, restrições genéricas e considerações de roteamento simples [17]. Contudo, apenas pode ser aplicado sobre determinadas condições de escalonamento e os resultados não são garantidos para problemas complexos.

Regras de prioridade

Um conjunto de regras foram definidas para resolução simples de problemas de escalonamento *job shop*, no que a planeamento de operações diz respeito. Das mais de 100 regras desenvolvidas, destacam-se:

- *Shortest Processing Time* (SPT) – É selecionada uma operação que tenha um tempo de processamento inferior relativo a um conjunto de operações;
- *First Come, First Serve* (FCFS) – É selecionada a operação que tem o maior número de etapas do conjunto de operações;
- *Most Work Remaining* (MWKR) – É selecionada a operação cuja operação pertence à ordem de produção com um tempo de processamento maior, tendo em conta as operações que ainda não foram planeadas;
- *Least Work Remaining* (LWKR) – É selecionada a operação cuja ordem de produção tem o menor tempo de processamento, tendo em conta as operações que ainda não foram planeadas.

Este tipo de regras são computacionalmente rápidas, no entanto a tentativa de melhoramento do *makespan* é muitas vezes limitado. Apesar dessa limitação são regras muitas das vezes a única opção em condições de escalonamento em tempo real [10].

2.1.3.4 – Abordagem baseada em Simulação

O uso da simulação em sistemas de manufatura apresenta resultados satisfatórios e demonstra ser uma boa ferramenta para uso diário de um chão de fábrica.

Vários estudos sobre regras de prioridade aplicadas em simulações computacionais foram realizados. Foram efetuadas várias pesquisas que descreveram o desempenho de regras de prioridade aplicadas em sistemas de manufatura recorrendo à simulação [18]–[23]. O desempenho das aplicações baseadas em simulação dependia de variados fatores, como o critério escolhido, a configuração do sistema e a carga de trabalho [24].

As ferramentas correntes, usadas para facilmente simular um chão de fábrica, tornam o planeamento e escalonamento uma tarefa mais simples. Usando a definição e aplicação de regras de prioridade para atribuição de ordens de trabalho, tendo em conta os recursos disponíveis, existe uma certeza que todas as restrições são consideradas e os objetivos de produção são satisfeitos. A abordagem de simulação tornou-se o ponto inicial para resolução de problemas de escalonamento, sendo que poderão ser construídas melhorias de *software*, para encontrar uma forma eficiente de reduzir o tempo de necessário para obter o escalonamento ideal [8].

2.1.3.5 – Abordagem baseada em *Machine Learning*

Meziane [25] introduz uma pesquisa sobre o uso de técnicas de Inteligência Artificial (IA) em sistemas de manufatura. Estes sistemas denominados como sistemas de manufatura inteligentes, são divididos nas componentes inteligentes, classificadas por *design*, planeamento de processos, gestão de qualidade, controlo, manutenção e diagnóstico e escalonamento. Sendo o escalonamento o tópico a tratar, é exposto escalonamento inteligente como sendo um problema de otimização de alocação de recursos sujeito a condições de sequenciamento e alocação. O objetivo da otimização consiste em alocar um conjunto de recursos a um conjunto de tarefas para que uma função de custo seja otimizada.

Das técnicas de IA usadas para obter um escalonamento inteligente destacam-se: Redes Neuronais (RN), Algoritmos Genéticos (AG), Lógica Difusa, Raciocínio Baseado em Casos e uma combinação entre as técnicas, denominados como Sistemas Híbridos. Também se tem em consideração uma abordagem seguindo algoritmos baseados em técnicas de *Reinforcement Learning*.

Redes Neuronais

O desenvolvimento de redes neuronais surge do conhecimento atual do sistema nervoso biológico, podendo imitar aspetos de processamento de informação dos humanos. Redes Neuronais são capazes de aprender a partir exemplos que funcionem como entradas e prever as saídas apropriadas, bem como generalizar relações entre elas.

A aplicação de Redes Neuronais a sistemas de manufatura é incrivelmente vasta, desde a fase de *design* até ao controlo, manutenção e escalonamento. Ainda assim existe um conjunto de problemas associados ao uso de Redes Neuronais:

- As decisões têm de ser feitas tendo em conta o tipo de Redes Neuronais a usar, a arquitetura, a topologia, o tipo de não-linearidade e os parâmetros associados;
- No caso de a aprendizagem ser do tipo *Supervised Learning*, é importante a seleção de um bom método de treino;
- Existe a necessidade de acelerar eficientemente o treino para garantir convergência rapidamente [26].

A complexidade computacional pode ser reduzida em problemas de escalonamento com o uso de Redes Neuronais. As Redes Neuronais conseguem apresentar resultados satisfatórios para resolução de problemas de otimização, através da implementação de um grande número de elementos com um elevado grau de conexão entre eles [25].

Algoritmos Genéticos

Algoritmos Genéticos (AG) usam uma população de soluções que conduzem a uma pesquisa robusta num vasto espaço de pesquisa. As estruturas representam as soluções candidatas, sendo inicialmente geradas aleatoriamente.

Cada estrutura é analisada pelo seu valor, ou *fitness*, dependendo da função de custo que se pretende otimizar. O algoritmo é gerado ciclicamente e cada iteração do ciclo é chamado de geração. Uma geração consiste em realizar a seleção e a recombinação. A seleção assume o papel de eliminar os piores e tentar alcançar valores cada vez mais próximos do ótimo. A recombinação pode ser dividida em duas fases, *crossover* e mutações. O *crossover* pode ser aplicado de variadas formas, em que por exemplo em dois cromossomas (c1 e c2) são escolhidos para a reprodução e são escolhidas aleatoriamente posições de c1 para trocar com c2, gerando uma descendência nova c3 que tem valores de c1 e c2, sendo estes valores ordens de produção. As mutações também poderão ser efetuadas por diferentes métodos. Pode por exemplo ser aplicada mudando completamente posições escolhidas aleatoriamente de um dado cromossoma. Nessas posições é alterada a sequência de ordens de produção e criado um cromossoma novo [27].

Candido [28] recorre a um Algoritmo Genético híbrido para resolução de problemas de escalonamento *job shop* com um grande número de restrições mais realistas. São consideradas ordens de produção com diferentes níveis, diferentes planos de processamento para as peças e recursos alternativos para as operações. Também é considerada a necessidade de múltiplos recursos para processar uma operação (máquinas, ferramentas, equipamentos, pessoal), calendarização de recursos, sobreposição de lotes e dependência de configurações das sequências. Candido usa o Algoritmo Genético aplicado a um conjunto de soluções ótimas locais, depois de obter um conjunto inicial de soluções. Min [29] usa um Algoritmo Genético aplicado a um escalonamento com máquinas idênticas em paralelo. É apresentado um algoritmo eficiente, obtendo soluções com qualidade superior às obtidas por procedimentos heurísticos ou o método de S*imulated Annealing*.

Lógica Difusa

Yu [30] descreve um escalonamento aplicado a um sistema de manufatura flexível (SMF) tendo em conta diferentes níveis. Cada critério ou objetivo tem preferência de nível no período de escalonamento, todos os objetivos são considerados, as alterações de condições da fábrica são captadas em tempo real por um modelo de inferência difusa que mapeia as alterações nos respetivos níveis e é implementado um escalonamento com multicritérios, em que é usado um método de partição baseado em níveis de preferência obtidos pela inferência difusa. O sistema proposto obteve um desempenho robusto no que diz respeito à carga de trabalho para todas as medidas de desempenho. O desempenho do sistema parece ser mais favorável quando a carga de trabalho é pesada.

Raciocínio Baseado em Casos

O Raciocínio Baseado em Casos (RBC) é implementado com base na experiência adquirida com soluções de problemas previamente resolvidos. Um Sistema Baseado em Conhecimento (SBC) adquire vantagens importantes em relação a outras abordagens, derivado a não necessitar de que as interações do problema sejam modeladas explicitamente. Cunningham e Smyth [31] demonstraram que podem ser produzidos escalonamentos satisfatórios para problemas de uma só máquina usando RBC. Escalonamentos anteriormente considerados bons podem ser considerados e parte deles pode ser reutilizada para uma construção rápida de novos escalonamentos. No entanto este método peca pela necessidade de adaptação do escalonamento ao problema, sendo necessário uma descoberta de métodos baseados em técnicas de reutilização para que se obtenham cenários de sucesso.

Técnicas de *Reinforcement Learning*

Numa abordagem com base em técnicas de *Reinforcement Learning* está presente um cenário em que um agente interage com um ambiente desconhecido, observa os resultados das suas ações e adapta o seu comportamento em função dos resultados. Gabel e Riedmiller [32] apresentam uma estratégia em que cada recurso é equipado com um agente, independente de outros agentes, que tomam decisões quanto às ordens de produção e em que é implementado um algoritmo *Reinforcement Learning* para melhoria da estratégia. Wang e Usher [33] analisaram os efeitos da aplicação de um algoritmo *Reinforcement Learning*, o *Q-Learning*, aplicado a um problema de máquina única, para seleção de regras de prioridade a aplicar. Neste algoritmo uma máquina, representado como um agente, escolhe uma de três regras, EDD (*Earliest Due Date*), SPT (*Shortest Processing Time*), FIFO (*First-In-First-Out*).

Sistemas Híbridos

Várias combinações entre métodos foram realizadas para tratamento de problemas de escalonamento *job shop*. Por exemplo uma combinação entre métodos de RBC e *Machine Learning* é capaz de reagir a problemas de escalonamento eficientemente [34]. Também uma combinação entre métodos de aprendizagem indutiva com Redes Neuronais pode ser aplicado a um escalonamento de um SMF. Kim, Min e Yih [35] extraem um conjunto de dados para construção de regras de escalonamento aplicando simulação e Redes Neuronais. Lee, Piramuthu e Tsai [36] apresentam uma combinação entre Algoritmos Genéticos e *Machine Learning* como sendo uma opção promissora para aplicações a problemas de escalonamento *job shop*. Neste método são lançadas ordens de produção ao longo do chão de fábrica usando técnicas de *Machine Learning* e é usado um Algoritmo Genético para atribuir as ordens a cada máquina.

Tendo abordado os principais métodos, adquire-se um especial interesse por técnicas de *Machine Learning* e de Simulação pela sua inovação e pelas vantagens associadas em comparação com outros métodos mais débeis.

2.2 – Machine Learning

Tendo em conta a análise feita previamente, é pertinente o estudo do tema *Machine Learning* de uma forma geral, dando destaque a subtemas à medida que sejam considerados

relevantes. Primeiramente são salientadas as diferentes aprendizagens associadas ao *Machine Learning* e seguidamente é abordado com mais detalhe o *Reinforcement Learning* bem como a sua aplicação para além da que já foi introduzida no ponto acima.

Li [37] define *Machine Learning* como sendo um conjunto amplo de métodos e algoritmos com a capacidade de aprender a partir de dados e de efetuar previsões e/ou decisões. *Machine Learning* usualmente é caraterizada em três tipos de aprendizagem, *Reinforcement Learning*, *Unsupervised Learning* e *Supervised Learning*.

Supervised Learning

Os algoritmos desenvolvidos desta técnica são definidos com a ajuda de um supervisor, responsável pelo conhecimento do ambiente e partilha com o agente a tarefa a executar. A função do algoritmo é transmitir um conjunto de dados com base na previsão e aprendizagem a partir de dados rotulados. Normalmente é usado para regressões e classificações.

Unsupervised Learning

No que diz respeito à aplicação de algoritmos desta técnica, não existe um conjunto de dados rotulados e tem como função encontrar padrões em dados de entrada e concretizar um agrupamento de dados. São construídos normalmente algoritmos normalmente usado para efeitos de *Clustering*.

Reinforcement Learning

Esta abordagem não possui um conjunto de dados rotulados, desempenhando a tarefa de aprendizagem com base no ambiente e conforme as decisões tomadas é concedido um feedback, uma recompensa que exprime a melhor decisão tomada. Em problemas do tipo *Reinforcement Learning* espera-se que o agente seja capaz de maximizar um acumular de recompensas [38].

Tendo analisado os diferentes tipos de aprendizagem previamente descritos, é realçado e estudado com detalhe o *Reinforcement Learning*, tendo em conta a sua relevância para o tipo de problemas a tratar.

2.2.1 - *Reinforcement Learning*

Reinforcement Learning (RL) consiste num método algorítmico, com capacidade de interação com o ambiente, desenvolvido com o objetivo de aprender uma política ótima e atingir um resultado desejado por tentativa e erro. Visando atingir o melhor resultado possível, não existe uma ação previamente definida, nem é conhecida qual a decisão que trará a maior recompensa. O melhor resultado é alcançado com base não só na recompensa atingida no momento, mas também na decisão que melhor suporta a decisão seguinte [38].

A aplicação de métodos *Reinforcement Learning* é vasta e pode ser enquadrada em diversas áreas, desde aplicações na Neurociência, Engenharia ou Matemática até aplicações mais detalhadas dentro de campos como *Machine Learning*, Controlo Ótimo ou Investigação Operacional.

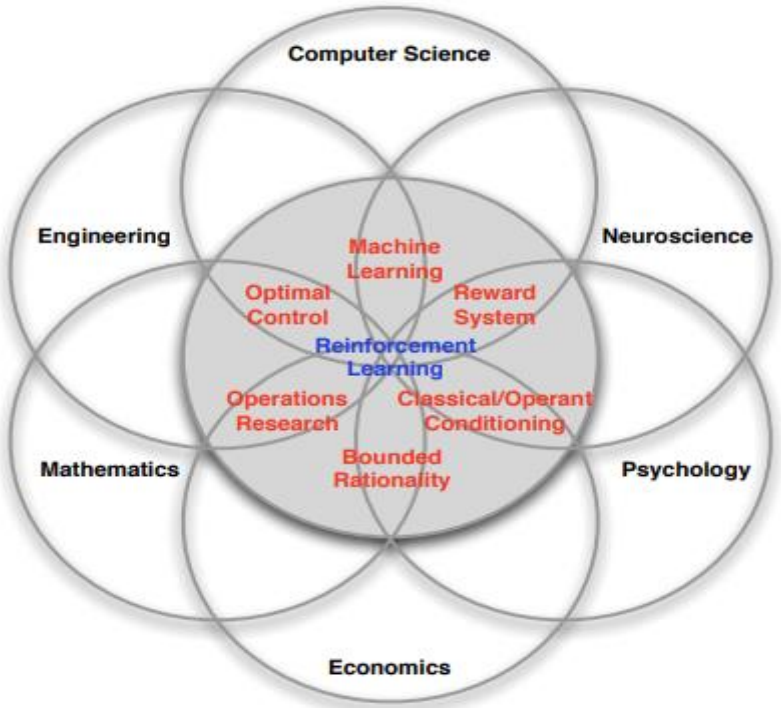

Figura 2.4 - Enquadramento do *Reinforcement Learning*, adaptado de [66]

Os métodos RL apontam para resolução de problemas em que as ações (decisões) são aplicadas a um sistema com um período de tempo extenso, com o fundamento de alcançar um objetivo desejado. Considerando que os métodos RL não carecem de um modelo do comportamento do sistema e em que as variáveis temporais são normalmente discretas, sendo as ações tomadas a cada passo, trata-se de um problema de *decision-making* sequencial. As ações são tomadas ciclicamente, o que conduz a que o resultado de ações anteriores seja monitorizado e considerado para a escolha de novas ações [39].

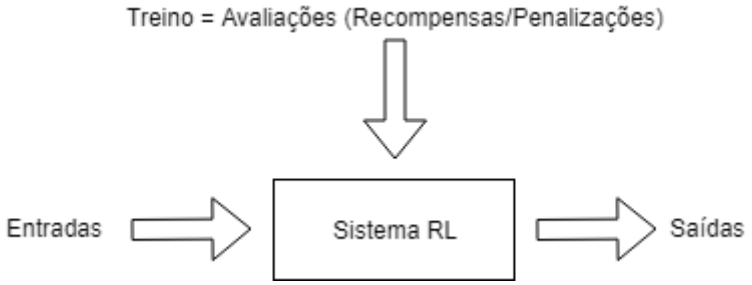

Figura 2.5 - Sistema *Reinforcement Learnig*, adaptado de [67]

2.2.2 – Caraterização de problemas *Reinforcement Learning*

Normalmente os problemas de *Reinforcement Learning* (RL) apresentam um conjunto de caraterísticas comuns, evidenciando-se o ambiente, o agente, a recompensa, a ação e o estado. Na figura 2.6 é demonstrada a conexão entre os vários componentes e de seguida será explicada a relevância de cada um.

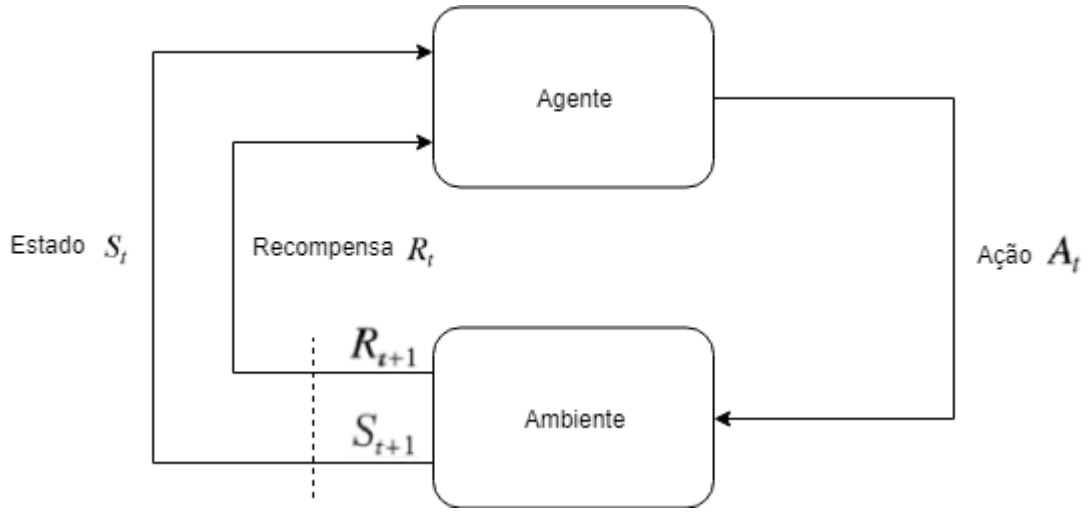

Figura 2.6 - Diagrama típico de problemas RL, adaptado de [38]

Agente

O agente representa o controlador do sistema, podendo observar o estado total ou parcial S_t do sistema. Ele interage com o ambiente executando ações A_t de acordo com o estado S_t e com a recompensa R_t recebida. O agente usa o estado e a recompensa para melhorar a sua política.

Ambiente

O ambiente corresponde a qualquer tipo de sistema que possa modelar uma determinada tarefa em problemas RL. É aqui que são definidos o estado e as ações disponíveis para o agente e implementa uma lógica onde são verificadas que ações mudam o estado atual. O estado do ambiente S_t pode ser discreto ou contínuo. A cada período t, o ambiente envia uma recompensa R_t para o agente, que serve de comparação para as ações tomadas em estados anteriores.

Ações

As ações representam a forma como o agente afeta o estado do ambiente. Elas podem ser discretas ou contínuas e a cada etapa podem ser tomadas mais que uma ação. Havendo a possibilidade de existir sequências de ações, o objetivo é encontrar a sequência que traduz a maior recompensa total.

Recompensa

A recompensa representa o principal objetivo a atingir em problemas RL. Ela descreve o imediato impacto que o agente recebe ao desempenhar uma dada ação sobre um certo estado do ambiente. Normalmente interessa considerar um conjunto de recompensas acumulativas, isto é, o acumulado de recompensas ao fim de um determinado tempo, visto que recompensas locais podem não atingir o valor máximo no futuro [40].

Política

Uma política π^* é definida como sendo o mapeamento de estados do ambiente às ações que estão a ser executadas no estado atual do ambiente. A política é formada por um conjunto de sequências de ações e reflete o comportamento do agente a um dado tempo. Como

normalmente o objetivo é otimizar o comportamento do sistema, para um determinado período de tempo, é determinada a política ótima para atingir um determinado objetivo global. A política é a base do comportamento do agente [38].

Face à importância do agente, é importante categorizar os variados tipos, tendo em conta o que se espera do agente. É importante referir que uma função de valor V^{π^*} corresponde à função ao longo do tempo do conjunto de políticas e um modelo surge como mapeamento de o que antecede e o que segue cada componente. Comum a todos os agentes está presente o mesmo objetivo, como encontrar uma política ótima. Como é percetível na figura imediatamente abaixo, um agente pode ser classificado pela função de valor, política e modelo.

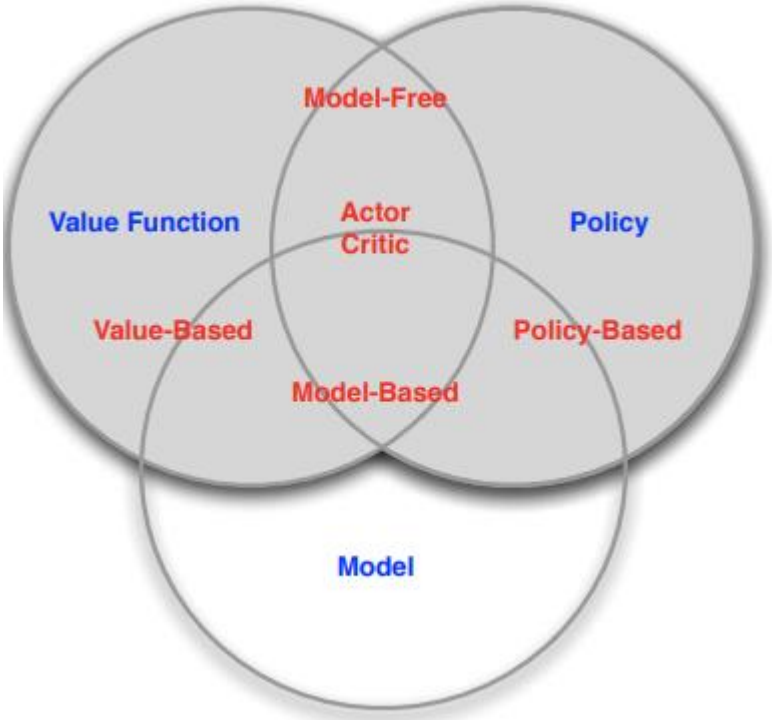

Figura 2.7 - Taxonomia de agentes RL, adaptado de [66]

Os métodos baseados na pesquisa de políticas mantêm a representação explícita das políticas π^* e modificam-nas através de operadores de pesquisa. Normalmente são usados métodos como programação dinâmica, valores de iterações, S*imulated Annealing* e Algoritmos Evolucionários. Por outro lado métodos baseados na pesquisa da função de valor tentam encontrar uma função V^{π^*} que retorne o máximo de recompensa acumulada. É esperado que o algoritmo aprenda a chegar às funções de valor através da experiência. A abordagem mais usada para este tipo de aprendizagem é o método da diferença temporal [41].

Uma vez compreendida a abordagem *Reinforcement Learning*, surgem as várias técnicas normalmente usadas para escalonamento, planeamento e controlo de sistemas.

2.3 – Técnicas de Reinforcement Learning

Neste ponto serão abordadas as técnicas *Neuroevolution* e *Q-learning*, bem como uma comparação entre elas, para que se possa compreender a sua usabilidade. Primeiramente o *Neuroevolution*, dividindo a técnica em duas partes, introduzindo o contexto de Redes Neuronais aplicadas a problemas de *Reinforcement Learning* e depois Algoritmos Evolucionários usados para criar e evoluir novas Redes Neuronais. Sobre o *Q-learning* irão ser abordados os fundamentos gerais e principais caraterísticas.

2.3.1 – *Neuroevolution* – Introdução

Neuroevolution é um método usado para modificar Redes Neuronais, mudando pesos, topologias, ou ambos, para aprender uma dada tarefa. Para maximizar a função de *fitness*, que mede o desempenho da tarefa, é usada computação evolucionária para pesquisa dos parâmetros da rede. Este método é bastante geral e permite uma aprendizagem sem que os objetivos estejam explícitos. *Neuroevolution* também pode ser visto como um método de pesquisa de políticas para problemas de *Reinforcement Learning*, adequado para domínios contínuos ou domínios onde o estado é apenas observável parcialmente [42].

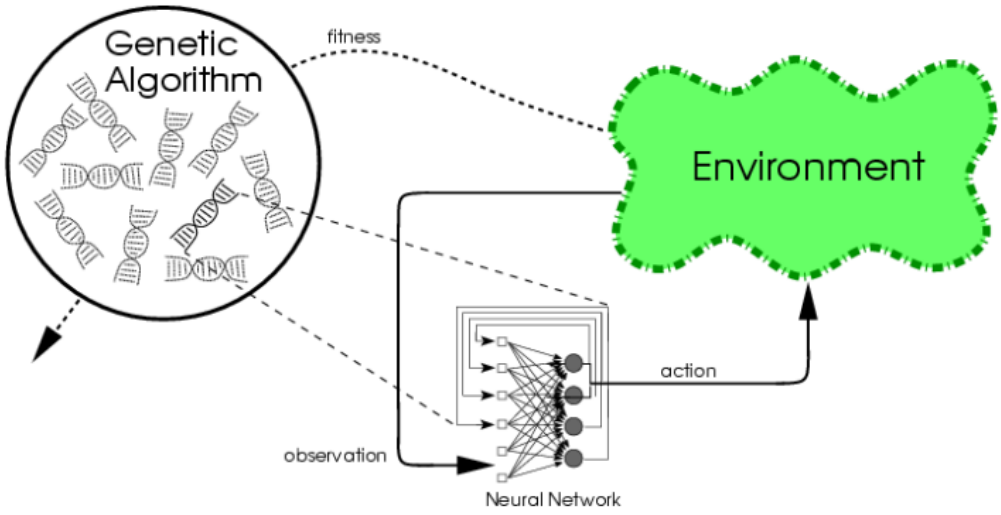

Figura 2.8 - Processo de evolução de Redes Neuronais, adaptado de [42]

Para uma melhor perceção de como funciona o método, é pertinente descrever o funcionamento geral tanto de uma Rede Neuronal Artificial, como do Algoritmo Genético, o algoritmo evolucionário usado pela NE para contruir e evoluir Redes Neuronais.

2.3.1.1 – Redes Neuronais

Ao longo dos últimos anos foram feitas pesquisas sobre modelos construídos a partir de Redes Neuronais Biológicas. Aplicando o conceito em várias áreas, destaca-se o uso de Redes Neuronais em Inteligência Artificial. Inteligência Artificial é definida como um campo dentro do âmbito da ciência dos computadores que se ocupa do estudo da automação de comportamentos inteligentes [43].

Redes Neuronais Artificiais podem ser vistos como modelos computacionais, implementados em *software* ou *hardware*, que se baseiam no comportamento e nas caraterísticas adaptativas

de um sistema nervoso biológico. Normalmente são compostas por unidades de processamento, apelidados de neurónios, interligadas entre camadas por conexões, denominadas de pesos da Rede Neuronal. O número e tipo de neurónios, bem como o conjunto de possíveis conexões entre eles define a arquitetura ou topologia da Rede Neuronal[44].

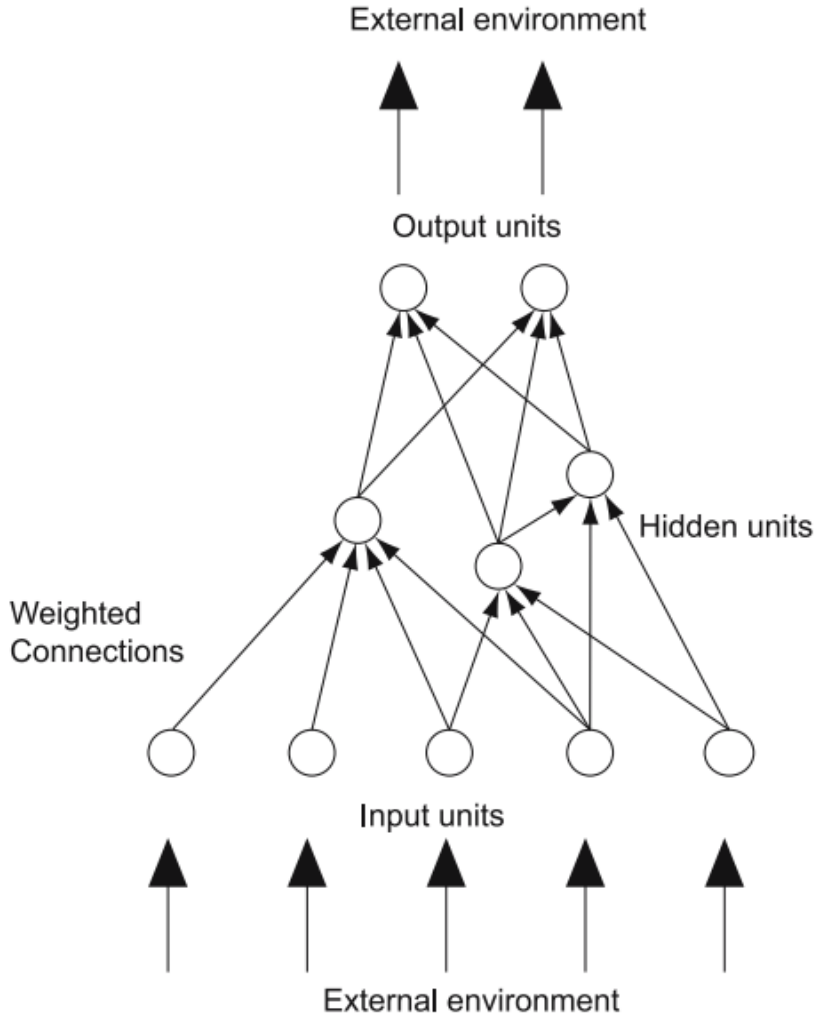

Figura 2.9 - Arquitetura genérica de uma Rede Neuronal, adaptado de [44]

Arquitetura

As Redes Neuronais Artificiais podem ser vistas como grafos direcionados, em que os neurónios artificiais são vistos como nós e os pesos como arestas ponderadas e direcionadas entre os neurónios de entrada e os neurónios de saída. A arquitetura pode ser agrupada em duas categorias, *feedforward* e recorrente.

- Redes com arquitetura *feedforward* – as conexões entre os nós não formam um ciclo. A direção é unidirecional, desde os nós de entrada até aos nós de saída.
- Redes com arquitetura recorrente – existe um ciclo entre a conexão dos nós. Este ciclo ocorre devida a conexões de *feedback*.

Na maior parte dos casos redes *feedforward* são estáticas, isto é, produzem um conjunto de valores dependendo de uma dada sequência de valores de entrada. As redes *feedforward*

não têm capacidade de memória, respondendo a um estado independentemente do estado anterior. Por outro lado, as redes recorrentes são vistas como sistemas dinâmicos. Derivado à existência de caminhos de *feedback*, as entradas são modificadas consoante o estado anterior.

Treino

O treino de uma Rede Neuronal é fundamental para obtenção de bons resultados. O processo de treino de uma rede pode ser visto como uma atualização da arquitetura da rede, bem como uma atualização dos pesos de conexão, para que seja desempenhada uma tarefa em específico. A base do treino é a relação entre entradas e saídas, não precisando de estarem definidas regras específicas, desenvolvidas por especialistas do sistema em questão.

Para que o treino seja eficaz, é necessário que esteja definido corretamente o modelo do ambiente em que as Redes Neuronais vão operar e que informação está disponível para a rede.

Também é importante ser considerado para o treino um peso denominado de *bias*. O peso do *bias* é um elemento que serve para aumentar o grau de liberdade dos ajustes dos pesos.

Existem três tipos de aprendizagem a considerar: S*upervised, Unsupervised* e Híbrida. Em *Supervised Learning* (SL) existe um supervisor responsável por fornecer as respostas corretas para um padrão de entradas. *Reinforcement Learning* é uma variante de SL em que não são fornecidas respostas mas sim uma crítica sobre a exatidão das saídas da rede. *Unsupervised Learning* (UL) por outro lado não possui nenhum tipo de resultados pré-definidos nem um supervisor. Quanto à aprendizagem Híbrida é resultado da combinação de SL e UL, em que parte dos pesos são normalmente determinados por SL, enquanto os outros são determinados por UL.

Funções de ativação

As funções de ativação usadas em Redes Neuronais podem ser vistas como formas de acelerar o treino, são responsáveis pelo processamento de cada neurónio. Conforme o tipo de Rede Neuronal são escolhidas as funções de ativação, podendo ser lineares ou não lineares.

De entre as várias existentes, as mais usadas habitualmente são: linear, linear com limites, sigmoide, tangente hiperbólica e *softmax*. A função linear é normalmente usada nas camadas de saída para regressões. Funções lineares com limites são usadas geralmente quando se pretende uma saída binária. A função sigmoide é mais usada quando se trata de Redes Neuronais com propagação positiva, isto é, com arquitetura *feedforward*, e onde se pretende uma saída entre 0 e 1. A tangente hiperbólica funciona no mesmo contexto mas é mais usada quando se pretendem saídas entre -1 e 1. Por fim, a função *softmax* é usada para classificações [45].

2.3.1.2 – Algoritmo Genético

Goldberg [46] define Algoritmos Genéticos como sendo algoritmos de pesquisa baseados em mecanismos de seleção e genética natural. Eles combinam a sobrevivência de estruturas com informação aleatória a um algoritmo de pesquisa com alguns princípios de pesquisa usada por humanos. A cada geração são criados novos segmentos usando partes dos segmentos anteriores.

Visto que os Algoritmos Genéticos são desenhados para simular um processo biológico, existem várias terminologias derivadas desse contexto ao qual é importante realçar o seu significado. É assim importante destacar: a função de *fitness* usada para otimização, a população de cromossomas, a seleção dos cromossomas que irão se reproduzir, o *crossover*

para produzir novas gerações de cromossomas e as mutações nos cromossomas implementadas em novas gerações [47].

Função *fitness*

A função *fitness* é a função que o algoritmo tenta otimizar. É usada para avaliar a qualidade das soluções, isto é, se uma solução é favorável ou não. Esta função é fundamental para o algoritmo pois é a partir da sua avaliação das soluções, que os cromossomas são descartados ou selecionados.

População de cromossomas

Como cromossoma entende-se como sendo um ou vários valores numéricos que representam uma solução candidata ao problema que o algoritmo está a tentar resolver. Uma população de cromossomas representa um dado conjunto de cromossomas. Normalmente, os Algoritmos Genéticos começam a primeira população com cromossomas aleatórios.

Seleção natural

A seleção natural visa escolher quais os cromossomas que irão sobreviver e fazer parte da próxima geração. Para a seleção são considerados os valores de *fitness* de cada cromossoma.

Crossover

O *crossover* é uma operação que permite a combinação de material genético de duas ou mais soluções. Pegando num exemplo simples da figura 2.10, a partir de dois ou mais pais são geradas descendências com informação de cada um deles.

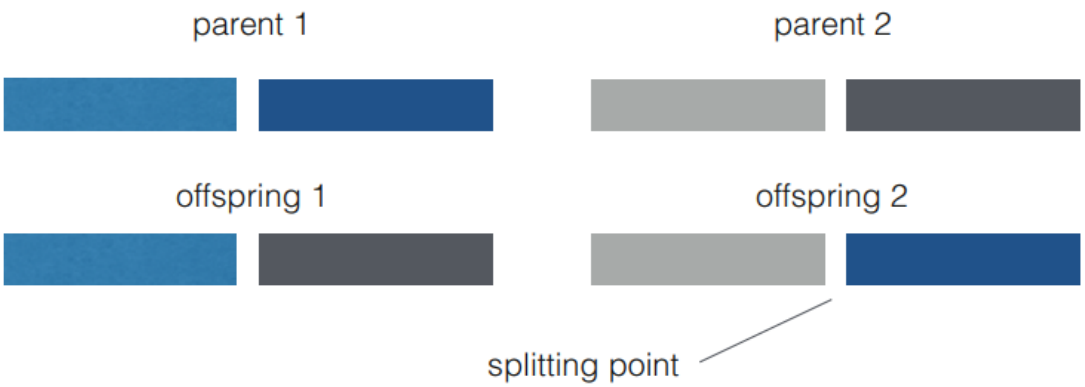

Figura 2.10 - Exemplo da operação *crossover*, adaptado de [48]

É importante considerar os pontos de *crossover*, isto é, pontos onde é dividida a informação recebida dos pais, podendo ser escolhidos de forma aleatória ou de forma uniforme. Existem vários métodos para realizar a operação de *crossover*. O *crossover* aritmético usa a média aritmética entre dois ou mais pares de solução para produzir uma descendência. O *crossover* dominante escolhe sucessivamente cada componente de um par de soluções. Por outro lado, o *crossover* uniforme é usado com base em uma taxa de seleção cada parte da solução.

Mutações

As mutações são usadas para que o algoritmo de pesquisa varie a área de pesquisa, na tentativa que encontre soluções globais ao invés de soluções locais. O processo das mutações ocorre depois do *crossover* e pode ser feito intencionalmente ou de forma aleatória em determinados pontos de um cromossoma [48].

2.3.2 – *Neuroevolution* – Implementação e aplicações

Depois de introduzida a abordagem *Neuroevolution* (NE), assim como os conceitos gerais de Redes Neuronais e Algoritmos Genéticos, é pertinente esclarecer como são usados Algoritmos genéticos para modificar e evoluir Redes Neuronais.

O objetivo da aplicação da metodologia *NE* é encontrar uma Rede Neuronal que melhor desempenhe uma tarefa específica. Para encontrar a melhor rede são criadas populações de codificação genética que possuem as várias Redes Neuronais (Genoma). Cada codificação da população (genótipo) é escolhida e decodificada na correspondente Rede Neuronal (fenótipo). A rede resultante é aplicada na tarefa, e é medido o seu desempenho ao longo do tempo, obtendo assim a função *fitness* para o correspondente genótipo.

Depois de criados todos os membros de uma população, são usados operadores genéticos para criarem novas gerações da população. Nas codificações com valor de *fitness* maior são aplicadas mutações e *crossover*. A descendência resultante é substituída nos genótipos com menor *fitness* da população. O processo de pesquisa é continuado até que seja encontrada uma rede com um valor de *fitness* favorável.

Numa abordagem convencional de *NE*, os pesos da Rede Neuronal são afinados e otimizados, tendo em conta uma Rede Neuronal com arquitetura fixa, o que torna o método fácil de implementar em vários domínios [42]. Outros tipos de abordagens de *NE* são usados quando é pertinente considerar que a modificação da topologia da Rede Neuronal é decisiva para obter melhores resultados. *Neuroevolution of Augmenting Topologies* (NEAT) é um método poderoso onde é demonstrado que, para além da modificação dos pesos da rede, a modificação da topologia pode constituir uma grande vantagem [49].

Várias aplicações, tanto da convencional NE como da NEAT, foram implementados em diversas áreas. Uma das aplicações é implementada para melhorar o processo de aprendizagem e de segurança da informação de estagiários numa empresa [50]. O algoritmo baseado em NE ajuda a esclarecer os níveis de aptidão e conhecimento existentes nos estagiários e ajuda a decidir quanto tempo o estagiário deve despender para cada nível de treino. Também aplicações do método em vídeo jogos foram feitas, como no videojogo Atari 2600 [51]. Neste caso é descrito e o método HyperNEAT como sendo um algoritmo sofisticado e com melhor desempenho que outros métodos de NE. Outras aplicações em tempo real foram feitas tanto em vídeo jogos [52], como em simuladores de carros de corrida [53].

2.3.3 – *Q-learning*

O *Q-learning* é um método de *Reinforcement Learning* (RL) proposto inicialmente por Watkins [54]. Tendo em conta a estratégia RL, em que é verificada a relação entre as ações implementadas num estado e a recompensa retirada desse estado, o algoritmo *Q-learning* faz a correspondência entre cada ação e um valor Q, sendo escolhida uma ação conforme este valor. O valor Q é definido tendo em conta as equações 2.1, 2.2 e 2.3, sendo que o valor estado-ação $Q^*(s,a)$ representa a soma de recompensas, tendo em conta um conjunto de políticas Π:

$$Q^*(s,a) = R(s,a) + \gamma \sum P_{ss'}^a . V(s', \pi^*) \quad (2.1)$$

$$V(s', \pi^*) = \max Q^*(s,a) \quad (2.2)$$

$$\pi^*(s,a) = \arg\max Q^*(s,a) \quad (2.3)$$

Onde, $R(s,a)$ é o reembolso obtido por uma dada ação dentro de um estado *s*. Derivado ao facto das funções P e R serem desconhecidas, $R(s,a)$ obtém o valor da função de estado-ação $Q^*(s,a)$ a cada iteração. O valor inicial Q pode ser escolhido intencionalmente e é atualizado segundo a fórmula 2.4 depois de executada uma ação de cada vez.

$$Q^*(s,a) = \begin{cases} (1-\alpha) Q_{t-1}(s,a) + \alpha \left[R(s,a) + \gamma max Q(s',a) \right], s = s_k, a = a_k \\ Q_{t-1}(s,a), outro \end{cases} \quad (2.4)$$

Explorando o espaço de estados, o valor Q tende para Q^* progressivamente. No entanto, quando se trata de estados de decisão largos, *Q-learning* não consegue trespassar todo o espaço [55].

Q-learning e Redes Neuronais

Face aos fundamentos de *Q-learning* apresentados no ponto anterior, é importante referir que este método também pode ser aplicado em conjunto com Redes Neuronais.

Deep Q-network (DQN) representa um agente que consegue combinar técnicas de *Reinforcement Learning* com uma classe de Redes Neuronais denominadas *Deep Neural Networks* (DNN), onde várias camadas de nós são usados para construir progressivamente uma maior representação de dados. Com DNN é possível compreender diretamente caraterísticas de objetos a partir de sensores [56]. Mnhi [57] desenvolveu um algoritmo onde aplica DQN ao videojogo Atari 2600.

Wang [58] combina Redes Neuronais com *Q-Learning* para melhoria de desempenho de um *robot* futebolístico. Neste caso, as diferentes variáveis do ambiente são as entradas da rede e os valores de Q que correspondem a ações são as saídas da rede. A recompensa é determinada por uma função de avaliação e o valor de Q corrigido a partir da recompensa obtida. O ajuste da Rede Neuronal é calculado com a diferença entre o valor de Q e os calculados pela rede.

Zhu [59] faz uso de Redes Neuronais para extrair caraterísticas para uma estrutura *Q-learning*, através de funções de aproximação e obtendo resultados semelhantes ao de DQN.

2.3.4 – Comparação entre os métodos

Vários algoritmos foram desenvolvidos e continuam a ser usados recorrendo a métodos de *Reinforcement Learning* como *Q-learning* e *Neuroevolution*. No entanto, a tendência para obtenção de resultados mais favoráveis assenta na ideia de usar Algoritmos Evolucionários para evoluir Redes Neuronais, como é o caso de *Neuroevolution*.

Petroski [60] destaca que com um simples algoritmo baseado em populações genéticas consegue resolver problemas complicados de *Reinforcement Learning.* Neste documento é realçado que o uso de Algoritmos Genéticos são a melhor opção para evoluir Redes Neuronais,

conseguindo obter melhores resultados e mais rapidamente que DQN, Pesquisa Evolucionária e A3C, método baseado em gradientes de políticas.

2.4 – Simulação de eventos discretos

Simulação de eventos discretos representa uma ferramenta poderosa para analisar e avaliar fluxos de produção. Estas ferramentas devem ser usadas como auxílio na melhoria da produção e podem ter um papel importante no dia-a-dia das empresas. As decisões que têm impacto na produção podem ser tomadas avaliando os processos sem ser necessário estagnar a produção, justificando assim o investimento.

A simulação é indispensável para resolução de problemas de muitos contextos reais. A Simulação é usada para descrever e analisar o comportamento do sistema e testar diferentes cenários [61].

Para melhor perceção de como é implementada a Simulação de Eventos Discretos é fundamental ter algumas noções base em consideração:

- Simulação – imitação de operações de processos ou do sistema em contexto real ao longo do tempo. A simulação envolve a geração de uma história artificial do sistema, e as observações dessa história servem para construir inferências, face às caraterísticas do sistema real representado.
- Modelo – representação do sistema real.
- Evento – representa uma ocorrência que muda o estado do sistema.
- Variáveis de estado – coleções de informação necessárias para definir o que está a acontecer no sistema a um dado período de tempo.

Também é importante ter em conta alguns conceitos comuns à maior parte das simulações. Isto inclui o sistema, modelo, eventos, variáveis de estado do sistema, entidades, atributos, atividades, atrasos e como é definido o modelo de simulação de eventos discretos.

Sistema, modelo e eventos

Um modelo é uma representação do estado atual do sistema e deve ser complexo na medida em que deve responder às questões pretendidas, mas não deve ser demasiado complexo. Sendo um evento uma ocorrência que muda o estado do sistema, um evento pode ser classificado em interno e externo, também apelidados de endógeno e exógeno. Um evento endógeno pode ser, por exemplo, o início do serviço a um cliente desde que esteja dentro do sistema simulado. Por outro lado, um evento exógeno pode ser visto como a chegada de um cliente para o serviço, uma vez que essa ocorrência está fora do sistema.

Existem vários tipos de modelos: matemáticos, descritivos, estatísticos, de entrada-saída e de eventos discretos. Um modelo de eventos discretos visa representar as componentes de um sistema e as suas interações, tendo em conta o objetivo de estudo. Os modelos de simulação discretos são dinâmicos, isto é, a passagem do tempo apresenta um papel crucial.

Variáveis de estado do sistema

As variáveis de estado representam a coleção de informação que define o que está acontecendo no sistema a um determinado nível num determinado tempo. A determinação de variáveis de estado depende de caso para caso, mesmo que representem fisicamente a mesma coisa, podem representar aspetos diferentes no modelo.

Entidades e atributos

Uma entidade representa um objeto que requere uma definição explícita. Uma entidade pode ser classificada como dinâmica, no caso de se mover com o sistema, ou estática, no caso de servir outras entidades. A cada entidade podem estar associados atributos caraterísticos, sendo considerados como valores locais. Um atributo de uma entidade pode ser por exemplo a cor da entidade.

Recursos

Um recurso representa uma entidade que fornece serviços a entidades dinâmicas, podendo servir uma ou mais entidades ao mesmo tempo, isto é, operar em paralelo. Uma entidade dinâmica pode precisar de uma ou mais unidades do recurso. Caso este esteja indisponível ou, a entidade é adicionada a uma fila de espera ou realiza outra ação. Caso haja disponibilidade e seja permitido a entidade permanece no recurso por um dado período de tempo e de seguida deixa o recurso.

Normalmente os estados dos recursos são cheio ou disponível mas também podem ser considerados outros, como em falha ou bloqueados.

Listas de processamento

As entidades são gerenciadas pela alocação a recursos, onde são feitas escolhas de quais as operações que devem esperar uma fila e quais devem ser processadas. Normalmente a fila é criada segundo a regra FIFO (F*irst-In-First-Out*), mas existem outras possibilidades de alocação.

Atividades e atrasos

Uma atividade corresponde à duração temporal em que é executada alguma tarefa. O período que demora a atividade é conhecido e pode ser escalonado. A duração temporal pode ser constante, aleatório resultante de distribuições estatísticas, resultado de uma equação ou baseado no estado de um evento.

Um atraso é uma duração indefinida causado por alguma combinação de condições do sistema. Simulações de eventos discretos contêm atrasos, por exemplo entidades em fila de espera. O início e o fim de uma atividade são considerados eventos.

Modelo de simulação de eventos discretos

Um modelo de simulação de eventos discretos pode ser definido pelas ações de variáveis de estado em pontos discretos de tempo onde ocorrem eventos. Os eventos ocorrem como consequência de atividades e atrasos. As entidades irão competir por sistemas de recursos e filas de espera, enquanto esperam por recursos disponíveis.

Uma corrida de simulação num dado período de tempo é feita por mecanismos que avançam o tempo simulado consecutivamente. O estado do sistema é atualizado a cada evento conforme são ocupados e libertados recursos num dado tempo [62].

2.4.1 – Vantagens da simulação de eventos discretos

A simulação de eventos discretos apresenta inúmeras vantagens:

- A simulação permite o estudo e experimentação de sistemas complexos;
- Permite testar eficazmente hipóteses sobre o como ou o porquê de um fenómeno ocorrer;

- Flexibilidade em manusear o tempo, podendo acelerar ou retardar períodos de tempo para analisar certos acontecimentos;
- Aquisição de conhecimento para melhoria do sistema;
- A avaliação de diferentes circunstâncias de simulação, alterando as entradas e verificando as saídas, pode produzir informações poderosas sobre o sistema;
- Podem ser testados novos desenhos de *hardware, layouts,* sistemas de transporte ou outras implementações no sistema, sem que seja necessário despender recursos com a sua aquisição;
- A simulação ajuda a formular e verificar soluções analíticas.

2.4.2 – Desvantagens da simulação de eventos discretos

Apesar das vantagens associadas ao uso de simulação de eventos discretos, também é importante considerar as suas desvantagens:

- É necessário especialização e treino para construir modelos de simulação;
- Uma vez que podem existir entradas aleatórias num modelo de simulação, uma observação pode ser resultado de aleatoriedades;
- Simulação e análise pode despender muito tempo e dinheiro [63].

2.5 – Considerações finais

Face aos tópicos abordados neste capítulo, torna-se como importante retirar breves conclusões sobre os vários temas abordados. As técnicas de *Machine Learning* demonstram um caráter inovador e tendem a obter melhores resultados que técnicas tradicionais e como tal aparentam ser uma solução interessante para sistemas de produção. Focando em mais detalhe em escalonamento da produção, no qual esta dissertação se insere, é importante realçar que técnicas capazes de aprender com base em interações com o ambiente, denominadas técnicas de *Reinforcement Learning,* demonstram ser técnicas adequadas ao contexto.

Face às vantagens do uso de Algoritmos genéticos para evoluir Redes Neuronais em comparação com outros métodos como *Q-learning*, é vista como boa opção a adoção desta técnica para resolver problemas de escalonamento. Não esquecendo o facto de haver necessidade de existir um ambiente para que o algoritmo possa ser testado e das claras vantagens de usar simulação de eventos discretos para otimização da produção, assume-se como favorável o uso de programas com capacidade de simularem contextos reais.

Capítulo 3

Abordagem metodológica

Face à dificuldade em simular e resolver um problema de escalonamento de um chão de fábrica, especialmente quando se tem em conta caraterísticas como tempos de processamento determinísticos ou *buffers* de *stock* intermédio de capacidade limitada, surge a necessidade de definir uma abordagem a seguir para obtenção de uma solução plausível e que faça sentido em contexto real.

Tendo em conta a revisão bibliográfica realizada no capítulo 2, a implementação de uma abordagem híbrida, baseada em técnicas de *Machine Learning* integradas com simulação, demostra ser uma abordagem adequada para obtenção de soluções viáveis num curto espaço de tempo.

Das diferentes abordagens existentes na área de *Machine Learning,* mais concretamente para problemas de *Reinforcement Learning* (RL), optou-se pelo uso do método *Neuroevolution* (NE), já abordado no capítulo 2 no ponto 2.3. Da pesquisa bibliográfica realizada, verificou-se que, mesmo não tendo sido implementado em casos de estudo com caraterísticas semelhantes, este método inovador demonstrou resultados satisfatórios em problemas de RL.

Depois de efetuada a pesquisa sobre simulação de eventos discretos no ponto 2.4 do capítulo 2, o uso do programa Simio apresenta uma solução bastante interessante para teste da qualidade de soluções de problemas de escalonamento de um chão de fábrica.

Na figura 3.1 está assente um diagrama geral da abordagem seguida onde está assente a conjugação de duas áreas: simulação de um chão de fábrica e o método NE.

Quanto à simulação, aplicada no Simio, são consideradas e simuladas caraterísticas e objetos de um chão de fábrica. Máquinas, operários, tarefas, tempos de processamento e ordens de produção são tidas em conta, tendo como objetivo simular exatamente um contexto real. A simulação consiste em testar diferentes cenários, em diferentes períodos de tempo, para que seja possível a visualização do melhor cenário possível.

A *Neuroevolution* é aplicado em API (*Application Programming Interface*). API permite ao utilizador alargar as funções do SIMIO, como construção de novas etapas, elementos e integração com algoritmos externos. Esta componente adquire a importância de ler entradas da simulação, como é o caso do número de peças em espera para processamento nas máquinas, e obter novas soluções para melhorar o escalonamento para um dado critério, como por exemplo, a produtividade.

Nas seções seguintes, serão discutidos com mais detalhe o funcionamento do programa Simio e a aplicação do método NE, para uma melhor perceção da abordagem seguida.

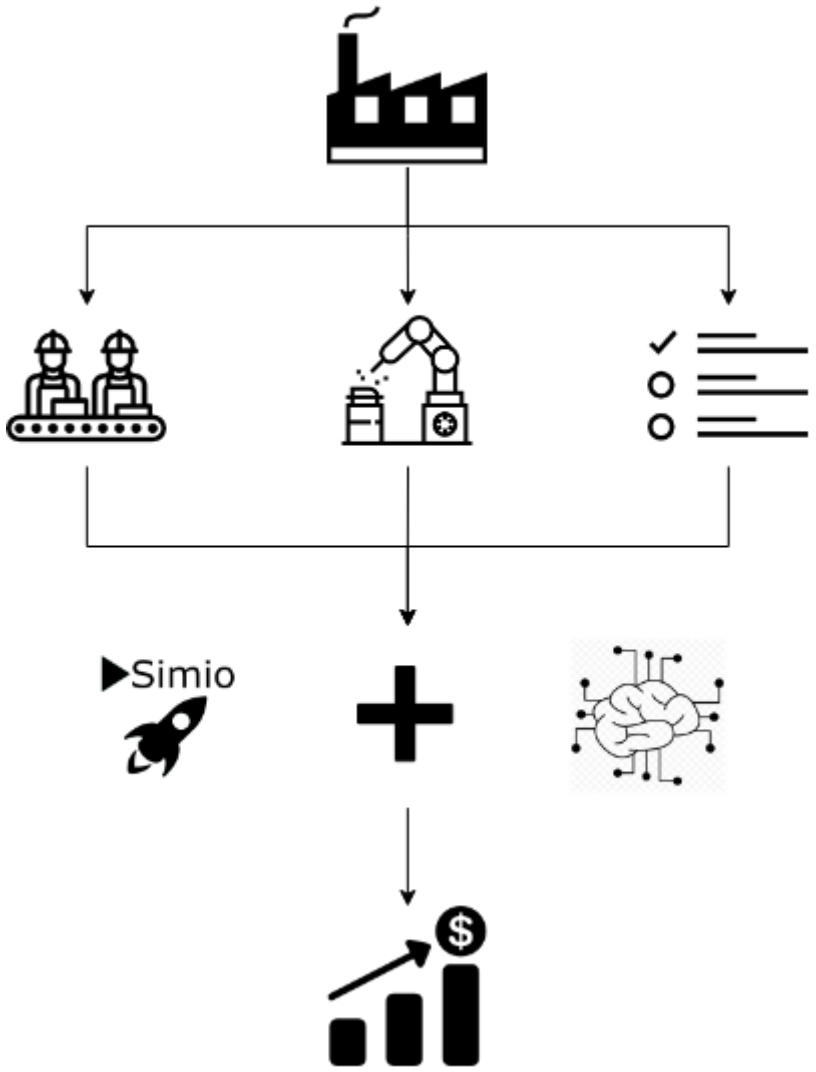

Figura 3.1 - Diagrama geral da metodologia usada

3.1 – Simulação de um chão de fábrica

A simulação é uma ferramenta bastante útil e aplicável em diferentes sistemas. Praticamente qualquer sistema de manufatura, produção ou de serviços pode ser simulado. Para além da simulação e melhoramento destes sistemas, o Simio tem a capacidade de auxiliar na resolução de problemas mais complexos. O facto de o Simio oferecer uma visualização 3D dos sistemas adquire uma vantagem fundamental em comparação com outras ferramentas. Com esta possibilidade, para além do responsável pelo desenvolvimento, também qualquer pessoa a quem seja facultado o modelo pode ver o sistema a funcionar em 3D [64].

Tendo em conta o contexto de simulação de chão de fábrica e para uma melhor perceção de como funciona o Simio, serão explicados alguns dos componentes fundamentais, bem como funções relevantes para a abordagem.

Modelo

Um modelo no SIMIO corresponde ao conjunto de posicionamentos entre os vários objetos, bem como o fluxo de entidades.

No contexto da abordagem seguida, o modelo no Simio serve para posicionamento de máquinas, trajetos, transportadores, pessoas e fluxos de peças. Também é visível no modelo a produção e finalização de peças.

As simulações do modelo podem ser controladas, parando sempre que seja pertinente analisar alguma situação. Pode também ser definido um tempo de simulação, ao qual o sistema de manufatura em contexto real estaria a ser executado. A visualização do modelo pode ser alternada entre 2D e 3D.

Na figura 3.2 é percetível um modelo 2D exemplo que representa um sistema de manufatura com três máquinas, um objeto de produção de peças e outro para finalização. Também vários comandos e objetos podem ser alterados pelo utilizador, como o tempo de início e finalização da execução da simulação.

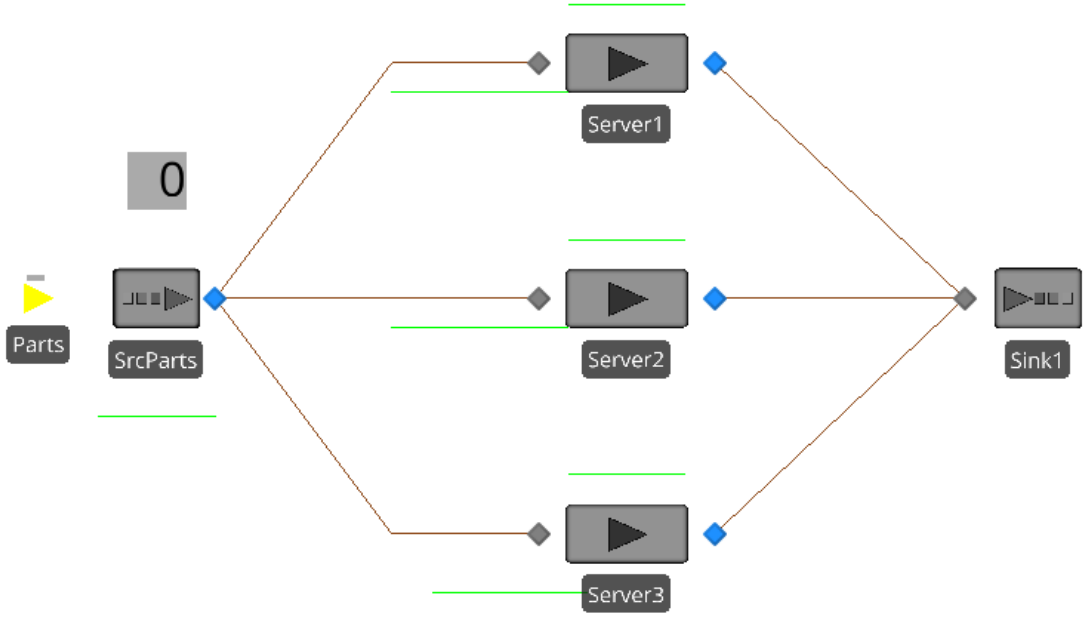

Figura 3.2 - Modelo exemplo no Simio com três máquinas e diferentes caminhos para processamento de peças

Entidade

Uma ou várias entidades são criadas selecionando um objeto “ModelEntity”. Uma entidade, por predefinição representada por um triângulo, adquire o papel de simular peças com determinadas caraterísticas. É assim possível, com apenas uma entidade representada, simular diferentes tipos de peças, podendo-se por exemplo alterar a sua cor e associar a cor a um tipo de peça. Na figura exemplo 3.2, a entidade “Parts” representa três tipos de peças diferentes, as quais irão ser produzidas e lançadas para produção no modelo.

Para a entidade criada é possível também a escolha de uma população associada, um roteamento e diferentes animações consoante a função que está a desempenhar em cada momento.

Servidor

Na metodologia seguida para simulação de um chão de fábrica um servidor representa uma máquina de trabalho, ao qual poderá estar associada uma tarefa específica de um contexto real, como cortar ou pintar. A cada servidor está associado por predefinição um *buffer* de entrada, representando peças em espera para processamento, uma área de processamento e um *buffer* de saída, onde são acumuladas peças já processadas. Para eventuais conexões estão também associados dois nós, um de entrada e um de saída. Tal como as entidades, também os servidores possuem um conjunto de propriedades acessíveis ao utilizador, como é visível na figura 3.3. Assim um utilizador pode definir quantas peças poderão ficar em espera, quantas podem ser processadas ao mesmo tempo, bem como quanto tempo demora a executar uma determinada tarefa.

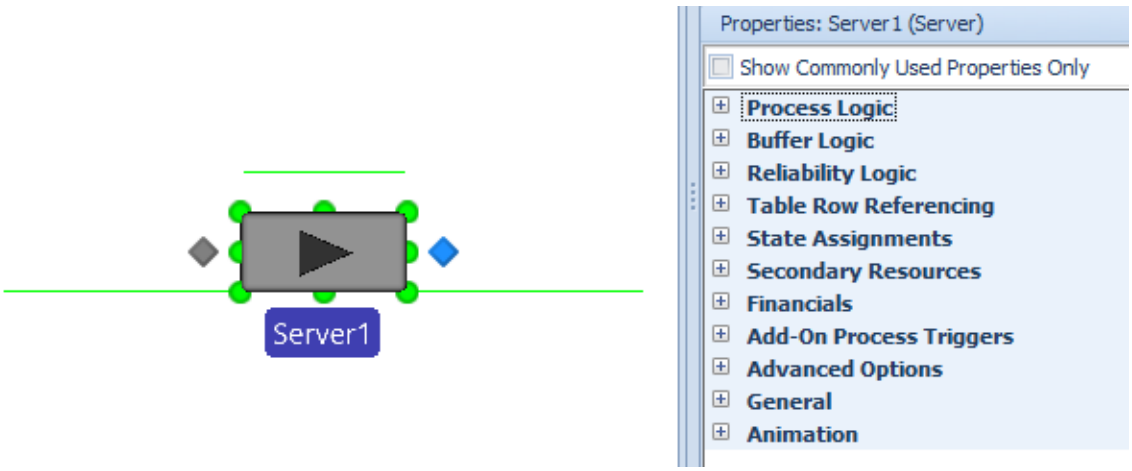

Figura 3.3 - Exemplo de um servidor num modelo do Simio e um conjunto de propriedades associadas

Trajeto

Um trajeto de um modelo do Simio representa o percurso a atravessar por uma peça, pessoa ou robot de um nó até outro. Para cada trajeto é possível controlar um conjunto de caraterísticas que melhor simulam um caminho real. Existem vários tipos de trajetos com diferentes caraterísticas. Por exemplo, um caminho temporal fornece ao utilizador a propriedade de escolha do tempo necessário para que uma peça atravesse o caminho completo.

Produção de peças

A produção de peças adquire um papel primordial na abordagem seguida, uma vez que será o local onde será efetuada uma das decisões do escalonamento pretendido. Para que se proceda à produção de peças é necessária a criação de uma "Source". Associada à "Source" está um *buffer* de saída, representando peças já produzidas e um nó de saída para possíveis conexões.

Tal como está presente na figura 3.4, à produção de peças está associada uma entidade e um conjunto de propriedades sujeitas a alterações por parte do utilizador, como o intervalo de tempo em que é produzida uma nova peça.

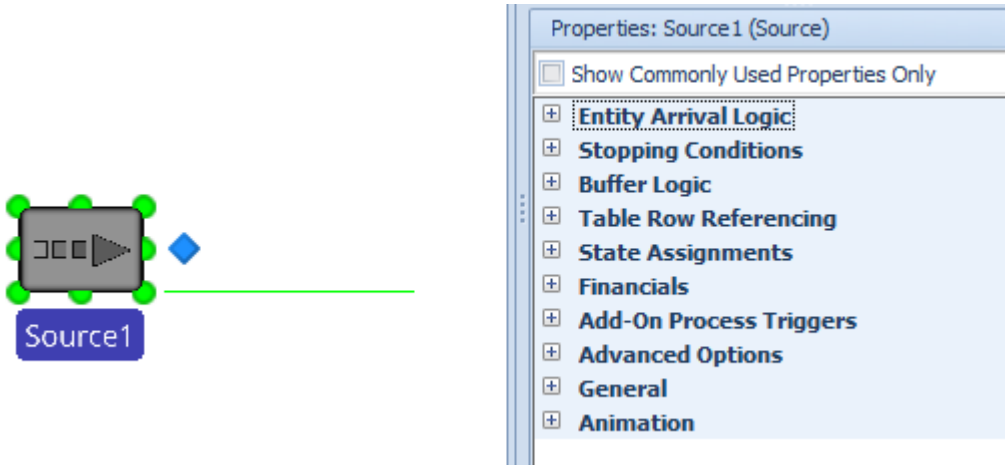

Figura 3.4 - Exemplo de um objeto para produção de peças e as propriedades associadas

Finalização de peças

Quando uma peça acaba todo o trajeto e é necessário que esta saia do sistema, é necessário uma "Sink". A "Sink", ao qual está associado um *buffer* de entrada e um nó, apresenta também um conjunto de caraterísticas suscetíveis a alterações por parte do utilizador, tal como é visível na figura 3.5.

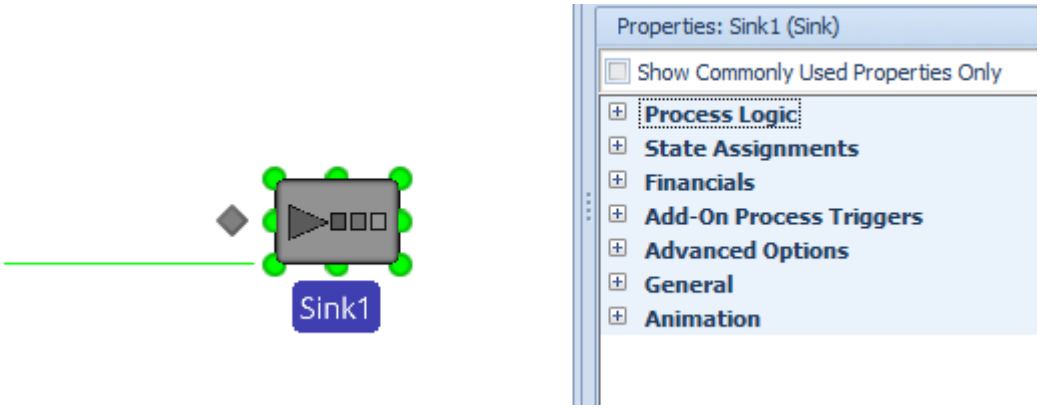

Figura 3.5 - Exemplo de um objeto para finalização de peças e as respetivas propriedades

Processos

Um processo no Simio tem como principal papel desempenhar funções executadas em segundo plano ao longo da simulação. Tendo em conta o objetivo do seu uso, um processo pode ser composto por passos e decisões, interligados a um início e a um fim. Sempre que um processo é chamado é acionado o início. Em cada processo podem ser representadas e executadas várias funções como atribuir, executar, encontrar, mover, como é visível no exemplo abaixo na figura 3.6. O utilizador também pode definir etapas externamente e usar em processos do Simio, a partir de API, tópico abordado posteriormente.

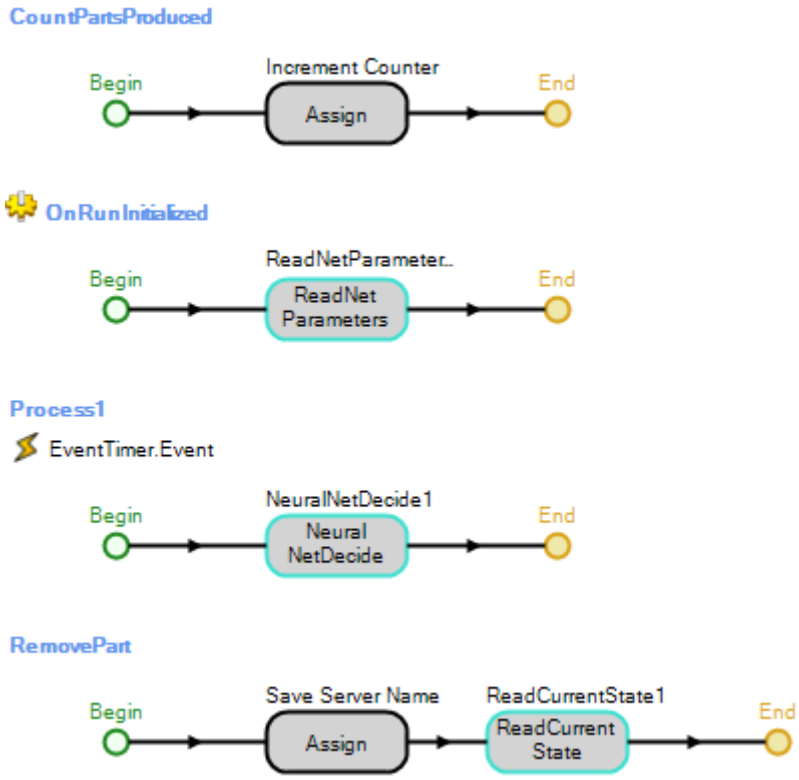

Figura 3.6 - Exemplo de processos com diferentes escolhas de etapas associadas

Definições

Ao longo da construção de um modelo é necessário definir novas variáveis para que sejam guardados valores, ou mesmo para acionar algum tipo de processo. Também propriedades, listas ou funções podem ser criadas nas definições. As variáveis e propriedades podem ser definidas de variados tipos, conforme a função que irão desempenhar e podem estar associadas tanto a um modelo como a uma entidade.

Dados

A necessidade de definição de caraterísticas usadas recorrentemente, como rotas constantes, tempos de processamento, ou outro tipo de dados a considerar, torna a definição de um conjunto de dados importante. A robustez do Simio permite que um conjunto de dados sejam inseridos tanto internamente como externamente, sendo apenas necessário usar a função *Bind* para que seja feita a conexão a um ficheiro externo como é o caso do Excel. Para a metodologia seguida os dados referentes ao tipo de peça, roteamento e tempos de processamento foram sempre definidos nesta secção.

Resultados

Tudo o que é simulado apenas faz sentido se forem verificados e analisados resultados. Sempre que uma simulação é pausada ou finalizada o Simio tem a capacidade de reportar resultados e estatísticas sobre vários objetos da simulação. Para além de filtros, para que sejam escolhidos apenas os objetos aos quais há interesse, também a estatística que se pretende

analisar é suscetível a alterações, podendo assim escolher um valor médio, um máximo ou outro tipo de estatística relevante.

Experiments

Tendo definido variáveis de controlo e respostas que funcionam como resultados de cada simulação, a partir de *experiments* é possível criar variados cenários de simulação. Para cada cenário também pode ser definido o número de replicações, importante quando existem variáveis estocásticas com variabilidade nas distribuições e intervalos de confiança.

Para a abordagem foi desenvolvida uma *experiment* customizada, sendo as variáveis de controlo pesos da Rede Neuronal, passadas por API e a resposta o número total de peças produzidas durante o intervalo de tempo definido, isto é, a produtividade.

API

Esta secção do Simio adquire papel fundamental na metodologia. A *Application Programming Interface* (API) permite ao utilizador alargar as funções do Simio, destacando-se:

- Construção de novas etapas, elementos e regras;
- Importação e exportação de dados;
- Construção de uma interface com o SIMIO a partir de programas externos;
- Integração com algoritmos externos.

Para que fosse possível integrar simulação de eventos discretos com Neuroevolution (NE), foi realizada uma conexão com um programa externo, o Visual Studio 2017, a partir de API. A partir deste programa externo, foi definida uma estrutura de dados que tem como função receber e enviar dados do Simio. Também várias etapas customizadas foram implementadas em API com o intuito de ler os dados correntes de simulação, ler os pesos da rede atuais, decidir instruções e carregar novos pesos.

A API também é usada para correr *experiments* sem que seja necessário abrir o programa diretamente. São enviados um conjunto de pesos como parâmetros de cada *experiment* e aplicada a metodologia NE, sendo extraídos os pesos da Rede Neuronal e os resultados da simulação.

3.2 – Neuroevolution (NE)

A A escolha do NE, previamente validada pelo capítulo referente à revisão bibliográfica, é uma parte fundamental na abordagem desenvolvida. Tratando-se claramente de um problema de *Reinforcement Learning*, o NE é implementado para construção de melhores Redes Neuronais usando Algoritmos Genéticos para possível otimização de um escalonamento de um chão de fábrica.

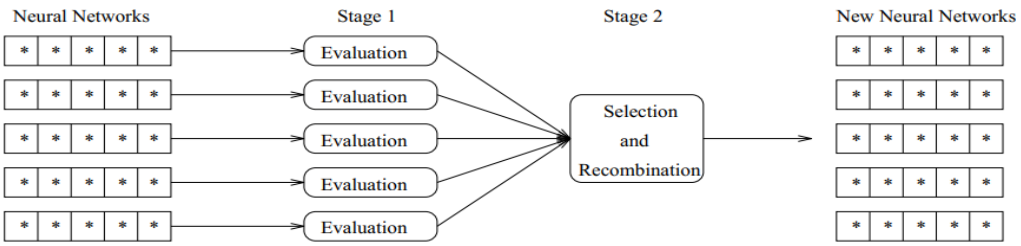

Figura 3.7 - Forma padrão da abordagem *Neuroevolution*, adaptado de [68]

Como é percetível na figura 3.7, será usada a abordagem NE, em que a criação de novas redes é feita através de um Algoritmo Genético. Tendo em conta a necessidade de integração deste método com o Simio, será explicado em pormenor o método e a sua aplicabilidade para a abordagem seguida.

3.2.1 - Redes Neuronais

Face à pesquisa realizada no capítulo 2 sobre Redes Neuronais, aplicadas na metodologia NE, foram exploradas Redes Neuronais que melhor se adaptem a problemas de escalonamento de um chão de fábrica. Optou-se por desenhar Redes Neuronais sem camadas ocultas, com apenas as camadas de entrada e saída, porque foi considerado que o problema não era complexo o suficiente para necessitar de camadas ocultas. O número de neurónios depende de problema para problema.

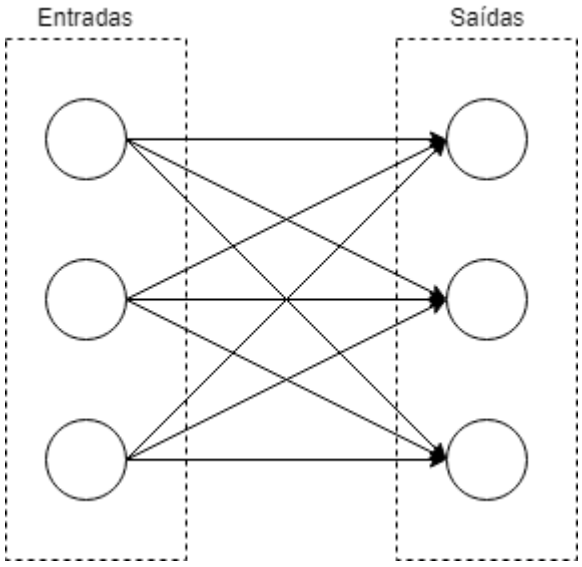

Figura 3.8 - Rede Neuronal exemplo com 3 neurónios de entrada e 3 neurónios de saída

Entradas

Sendo a camada composta por vários neurónios, na abordagem seguida, cada neurónio de entrada representa o número de peças de cada tipo em espera para processamento em cada máquina. Ou seja, seguindo a figura 3.8 como exemplo, o primeiro neurónio poderá representar o número de peças do tipo 1 em espera para processar na máquina 1.

Pesos

Os pesos da Rede Neuronal entendem-se como sendo valores numéricos resultantes da ligação entre neurónios de camadas consecutivas. Utilizando como referência mais uma vez a figura 3.8, tendo 9 ligações entre os neurónios de entrada e saída, a Rede Neuronal possui um total de 9 pesos entre os neurónios. Por simplicidade, foram sempre considerados valores para os pesos entre -1 e 1.

Para além dos pesos considerados entre os neurónios, também é importante referir a existência do peso de cada *bias*. Sendo a sua função o ajuste dos pesos, apenas existe peso de *bias* depois da camada de entrada. Na figura 3.8, apesar de não estarem representados, seriam

considerados 3 pesos de *bias*, um por cada neurónio de saída, ficando com um total de pesos entre entrada e saída de 12 pesos.

Saídas

Cada neurónio da camada de saída representa o tipo de peça na abordagem. Após aplicada a função de ativação a cada neurónio da camada de saída, um dado valor entre 0 e 1 é retornado. Escolhendo o máximo entre os valores de cada neurónio, é determinada o tipo de peça escolhido para processamento.

Função de ativação

O processamento em cada neurónio é feito pela função de ativação. Sendo a escolha de uma função de ativação uma consideração importante, optou-se por escolher a função sigmoide. A escolha deve-se ao facto de ser correntemente usada por redes neuronais com propagação positiva (*feedforward*), como é o caso, e também porque cria um *output* com um valor numérico entre 0 e 1, o que permite definir à partida o intervalo de valores que a rede vai produzir.

3.2.2 – Algoritmo Genético

Depois de bem definida a estrutura da Rede Neuronal, a abordagem NE baseia-se em aplicar Algoritmos Genéticos para que uma dada função de custo seja otimizada. Para usar Algoritmos Genéticos são construídos genomas compostos por informação proveniente da Rede Neuronal. O genoma é construído ordenadamente pelo conjunto de ligações que chegam a cada neurónio da camada de saída. Assim, como é percetível na figura 3.9, w4 representa um vetor de pesos com todos os pesos que ligam ao neurónio 4, incluindo o peso de *bias* e o mesmo se aplica para w5 e w6.

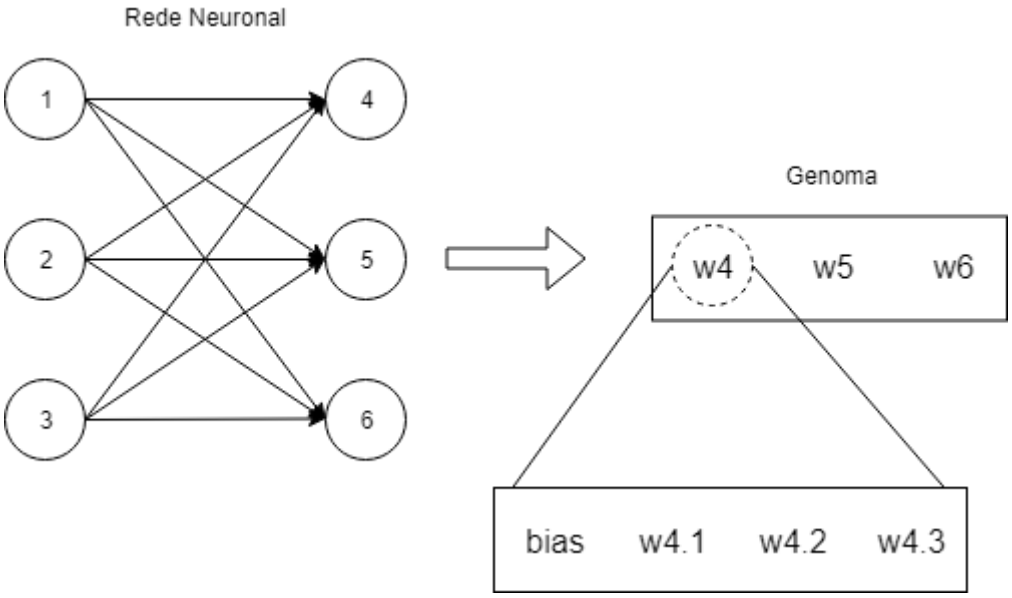

Figura 3.9 - Exemplo de como é construído um genoma a partir de uma Rede Neuronal

Sendo cada genoma um conjunto de vetores de pesos, estando associadas todas as ligações à camada de saída, foi aplicado um Algoritmo Genético com o objetivo de encontrar um genoma que melhor desempenhasse a função pretendida.

Depois de definidos quais são os parâmetros do Algoritmo Genético e de escolhida uma função de custo adequada, a figura 3.10 apresenta o processo geral de como é feita a sua aplicação.

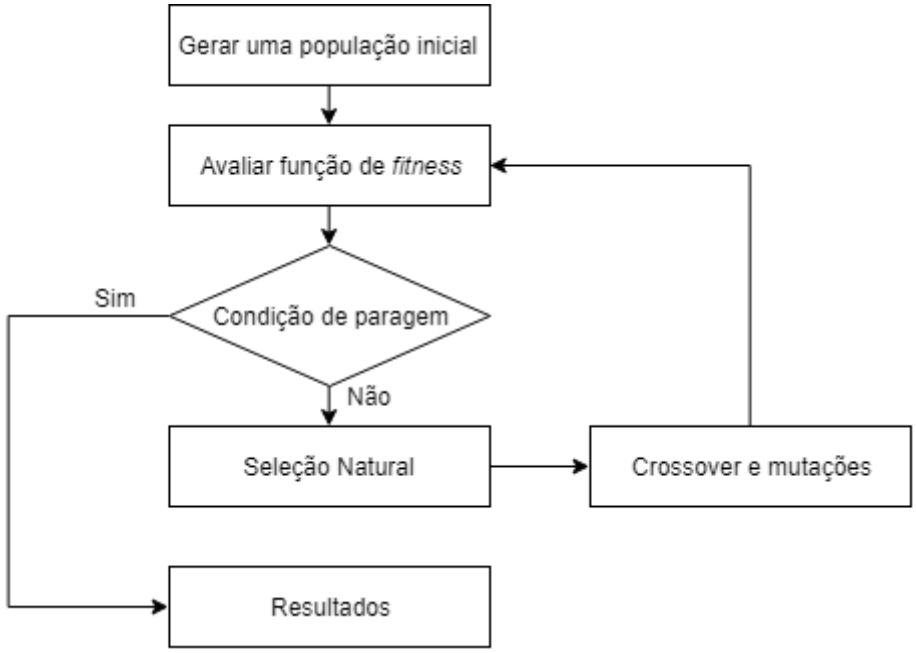

Figura 3.10 - Diagrama geral da aplicação do Algoritmo Genético, adaptado de [69]

População inicial

Como população inicial são criados um conjunto de pesos de Redes Neuronais. No contexto da abordagem seguida, foi implementada uma aleatoriedade de pesos na construção de cada Rede Neuronal, estando cada peso compreendido entre -1 e 1.

Função de custo

Depois de gerada uma população de redes, é avaliado o resultado gerado (*fitness*). Na metodologia é considerada a produtividade como sendo a função de *fitness*.

Seleção natural

Na seleção natural são avaliados os valores de *fitness* de cada rede e escolhidos os melhores para sobreviverem, podendo possivelmente reproduzirem descendência na geração seguinte. Para criação de uma nova população são selecionados os melhores, provenientes da população anterior e da seleção dos melhores cromossomas após *crossover* e mutações. Da figura imediatamente abaixo, figura 3.11, está visível um exemplo de como pode ser realizada a seleção, onde cada círculo numerado representa todos os vetores de pesos de uma Rede Neuronal.

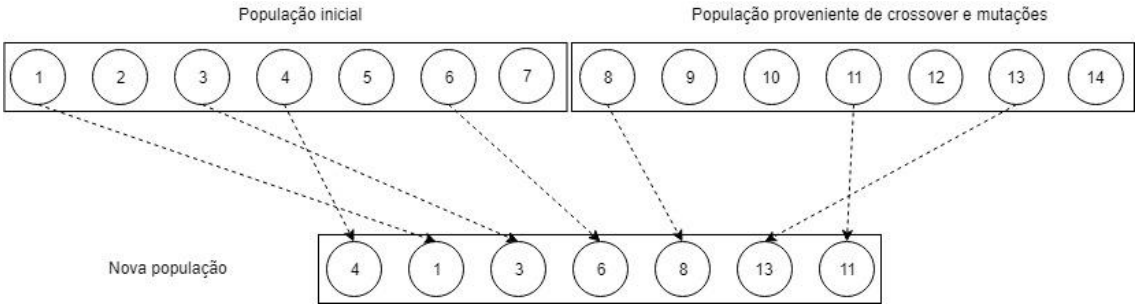

Figura 3.11 - Exemplo da seleção natural implementada, cada círculo numerado representa uma Rede Neuronal diferente

Acasalamento *e Crossover*

Para acasalamento são escolhidos aleatoriamente dois descendentes para se reproduzirem. No *crossover* é realizada a troca de informação entre eles, produzindo um filho com partes de cada um deles. De entre os variados métodos de *crossover* existentes, optou-se por trocar os pesos que estavam ligados ao mesmo neurónio de saída [65]. Sendo um cromossoma o conjunto de vetores de pesos de uma Rede Neuronal, na figura exemplo 3.12 foi criada uma descendência em que se mantiveram a maior parte dos pesos do cromossoma 2, mas um vetor de pesos ligados ao mesmo neurónio foi trocado pelo equivalente do cromossoma 1. É gerada uma população com número de Redes Neuronais igual à inicial, sendo posteriormente aplicadas mutações em 40% dessa população.

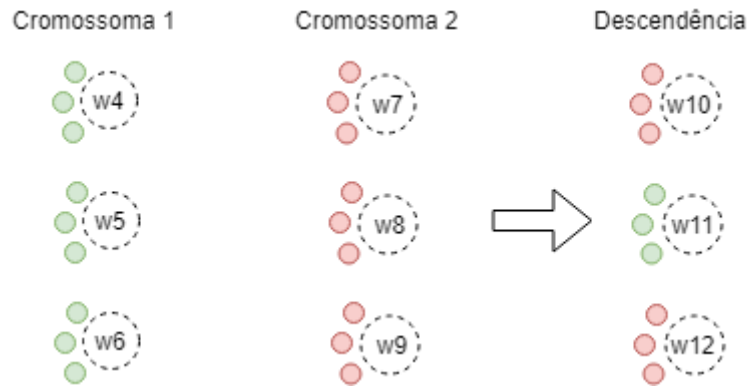

Figura 3.12 - Exemplo de *crossover* entre dois cromossomas. Cada círculo colorido representa os pesos que fazem parte de cada vetor de pesos w

Mutações

As mutações adquirem o papel de pesquisar em regiões fora da área de tendência para que a pesquisa não fique estagnada em um mínimo local. Depois de aplicado o *crossover* são substituídas partes do cromossoma por novas, isto é, são substituídos pesos por outros diferentes dos pais em pontos escolhidos para mutação. Tendo em conta os vários métodos de mutações existentes, optou-se por escolher aleatoriamente pontos de mutações dentro de um conjunto de pesos de cada Rede Neuronal. Os pontos de mutação escolhidos de forma aleatória estão compreendidos entre 1 e 20% do número total de pesos.

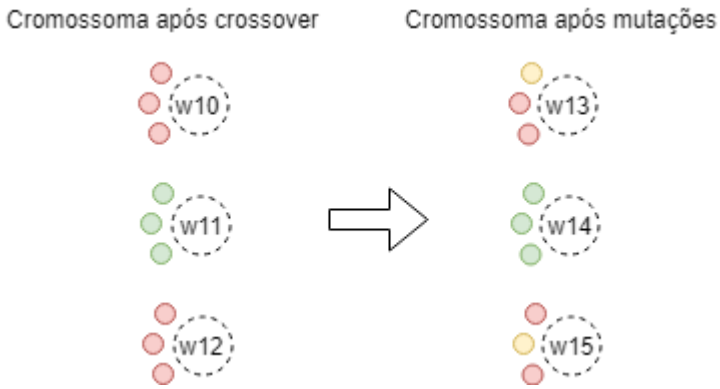

Figura 3.13 - Exemplo de mutações, com 2 novos pesos gerados aleatoriamente após crossover

Obtenção de resultados

Após verificação de que o valor de *fitness* não se alterou significativamente após um conjunto de iterações, é definido o valor final para o qual convergiu o algoritmo.

3.3 – Combinação de *Neuroevolution* com simulação de eventos discretos

Uma vez explicado no ponto 3.1 o funcionamento do Simio e no ponto 3.2 a descrição do método NE, é pertinente compreender a sua conexão para que a abordagem seja robusta e eficaz. Na figura 3.14 é representada a metodologia aplicada a um diagrama típico de problemas de *Reinforcement Learning*. O Simio, usado para simular um chão de fábrica e testar diferentes cenários, representa o ambiente, enquanto as Redes Neuronais representam o agente.

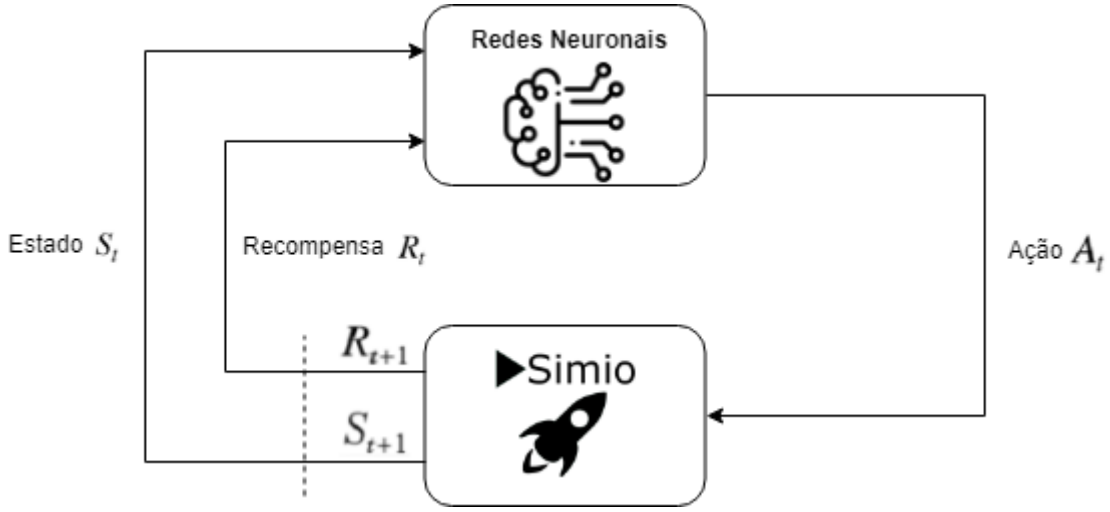

Figura 3.14 – Aplicação do método NE e Simio em problemas *Reinforcement Learning*

Como é percetível na figura 3.13, as Redes Neuronais atuam em conformidade com a recompensa obtida e são influenciadas pelo estado anterior no Simio. O Algoritmo Genético tal como referido no ponto 3.2 surge com o papel de melhorar as Redes Neuronais, para que sejam capazes de atingir melhores recompensas.

Por fim, tendo sido desenhada a abordagem metodológica a seguir, serão conhecidos quais os casos de estudo a que se aplica, bem como os resultados alcançados.

Capítulo 4

Caracterização dos Casos de Estudo

O presente capítulo tem como objetivo descrever todas as caraterísticas relevantes consideradas em cada caso onde foi aplicada a abordagem. Serão descritos casos de estudo distintos, sendo importante explicar as caraterísticas que diferenciam cada um dos casos.

Os diferentes modelos de simulação realizados no Simio adquirem o papel de versão simplificada de problemas reais de um chão de fábrica com que o INESC TEC lida. A estrutura da Rede Neuronal é caraterizada com base na revisão bibliográfica feita no capítulo 2 e é diferente de caso para caso, derivado das caraterísticas de cada um deles.

Cada caso de estudo foi pensado tendo em conta o nível de complexidade que se pretendia e se aplica no dia-a-dia de um sistema de produção. Primeiramente irá ser descrito um caso mais simples, seguido de um mais complexo, onde é aumentado o número de componentes constituintes do chão de fábrica e por fim será descrito um caso de estudo real, onde será considerado o modelo de simulação de uma empresa de semicondutores.

É pertinente referir que, depois de caraterizados os casos de estudo, foram construídos cenários de teste, para que seja analisada a metodologia seguida. Tendo em conta os casos de estudo que serão apresentados, cada cenário foi desenvolvido com a preocupação de que as caraterísticas se assemelhassem a ambientes de fabrico reais. Para avaliação da eficiência da metodologia e comparação com regras de prioridade foram criados cenários estratégicos, com evidências de estrangulamentos de produção e diferenças consideráveis entre tempos de processamento e capacidades das máquinas de produção.

4.1 – Caso de estudo nº1

O primeiro caso de estudo assume-se como o mais simples, onde é representado um chão de fábrica com três diferentes tipos de peça a serem processadas em três máquinas, não havendo partilha de recursos. Na figura 4.1 é visível um esquema geral do caso de estudo. O escalonamento será principalmente determinado pelo facto de apenas existir uma possibilidade de processamento para cada peça. O caso será caraterizado pelo tipo de peças, o lançamento de peças, processamento e finalização.

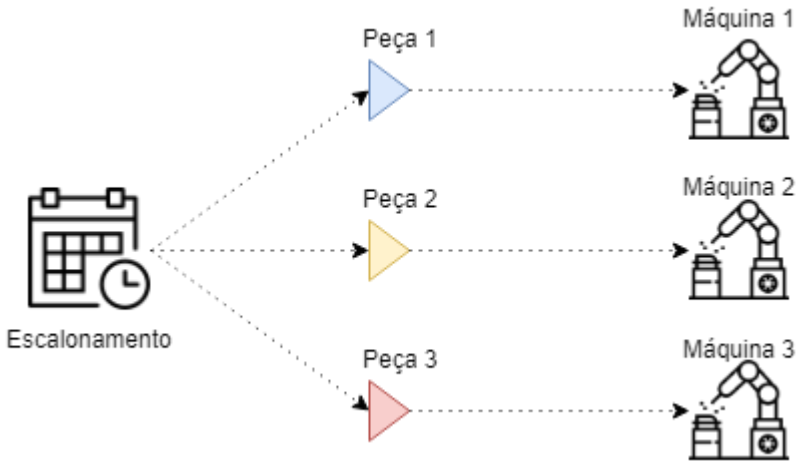

Figura 4.1 - Esquema geral do caso de estudo nº1

Tipo de peças

O sistema em estudo possui três diferentes tipos de peça, com diferentes atributos, nomeadamente, cor e tempos de processamento nas máquinas.

Lançamento de peças

O lançamento de peças é visto como um processo de escolha de peças a serem processadas. Estando todas as peças disponíveis para produção e não existindo limite de *stock*, é determinada a peça que irá ser lançada num determinado tempo. O tempo de lançamento de peças para o sistema é constante e igual a 1,5 minutos, isto é, a cada 1,5 minutos é lançada uma nova peça. Em diferentes cenários este tempo pode ser modificado para que sejam enviadas peças mais ou menos regularmente, tendo em conta se se pretende acumular mais ou menos peças nos *buffers* de entrada das máquinas.

Processamento

O processamento das peças é efetuado em três máquinas. Cada máquina apenas processa uma peça de cada vez e cada máquina é responsável por um tipo de peça. Na tabela 4.1 são descritos os tempos de processamento inicias de cada peça em cada máquina, representando apenas o tempo que estão efetivamente em processamento sem contabilizar tempos de espera nos *buffers*. Estes tempos serão modificados posteriormente para análise de diferentes cenários e testar o impacto da alteração dos tempos de produção na abordagem.

Tabela 4.1 - Tempos de processamento em minutos de cada peça para o caso de estudo nº1

	Máquina 1	Máquina 2	Máquina 3
Peça 1	5	-	-
Peça 2	-	4	-
Peça 3	-	-	4

Antes e depois de processarem, as peças são inseridas em *buffers* segundo a lógica FIFO (*First-In-First-Out*), isto é, a primeira peça a chegar ao buffer é a primeira que irá ser processada. Para este primeiro caso de estudo apenas é importante referir os *buffers* de

entrada, uma vez que a peça passa diretamente do processamento para a finalização sem tempo de espera. A capacidade de cada *buffer* de entrada de cada máquina é limitada e inicialmente os seus valores são os da tabela 4.2. Tal como para os tempos de processamento, estes valores de capacidade serão alterados para testes de diferentes cenários e medir o seu impacto.

Tabela 4.2 - Capacidade do *buffer* de entrada em cada máquina para o caso de estudo nº1

	Capacidade
Máquina 1	5
Máquina 2	5
Máquina 3	5

Finalização

Como finalização entende-se como sendo o acabamento de todas as operações numa peça. Neste caso de estudo, as peças são consideradas finalizadas depois de processadas nas máquinas e são enviadas para armazém para serem contabilizadas e não voltam a entrar no sistema de produção.

Para o presente caso de estudo também se assume relevante considerar a estrutura da Rede Neuronal, uma vez que é variável de caso para caso. Apenas serão descritas as caraterísticas relevantes e a estrutura, uma vez que o seu tratamento foi descrito na abordagem metodológica no capítulo 3. A topologia da rede é a apresentada na figura 4.2, com apenas duas camadas, entradas e saídas, cada uma com três neurónios.

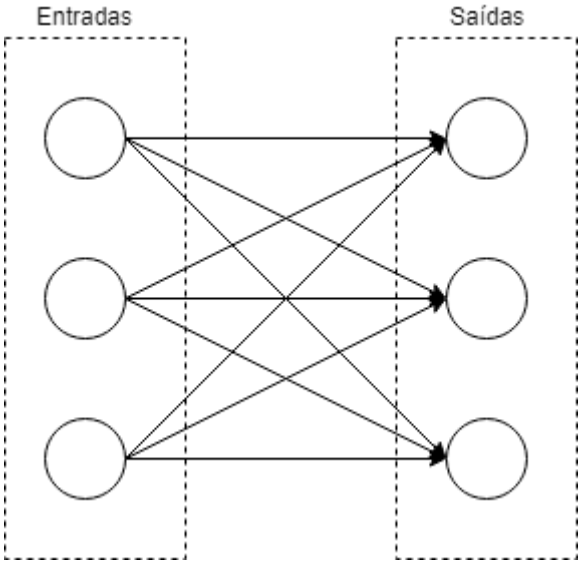

Figura 4.2 - Estrutura da Rede Neuronal para o caso de estudo nº1

Cada neurónio de entrada representa o número de peças de cada tipo em cada *buffer* de entrada das máquinas. Assim, o primeiro neurónio representa o número de peças do tipo 1 no

buffer da máquina 1, o segundo o número de peças do tipo 2 no *buffer* da máquina 2 e o terceiro as peças tipo 3 na máquina 3. Uma vez que cada tipo só é processado numa máquina apenas interessa considerar três neurónios.

Os neurónios de saída representam o tipo de peça a ser lançada. Cada neurónio representa um tipo de peça diferente. Também é importante considerar os pesos de *bias*, um por cada neurónio de saída, ficando a Rede Neuronal com um total de 12 pesos.

4.2 – Caso de estudo nº2

Um segundo caso de estudo foi considerado para aumentar a complexidade do escalonamento. Este caso de estudo é um chão de fábrica com 5 diferentes tipos de peças a serem processadas em 7 máquinas, com tempos de processamento e sequenciamento diferentes, tratando-se claramente de um problema *job shop*. O trajeto que cada peça atravessa até estar finalizada está presente na figura 4.3. Mais uma vez o escalonamento será principalmente caraterizado pelo tipo de peças, o lançamento de peças, processamento e finalização.

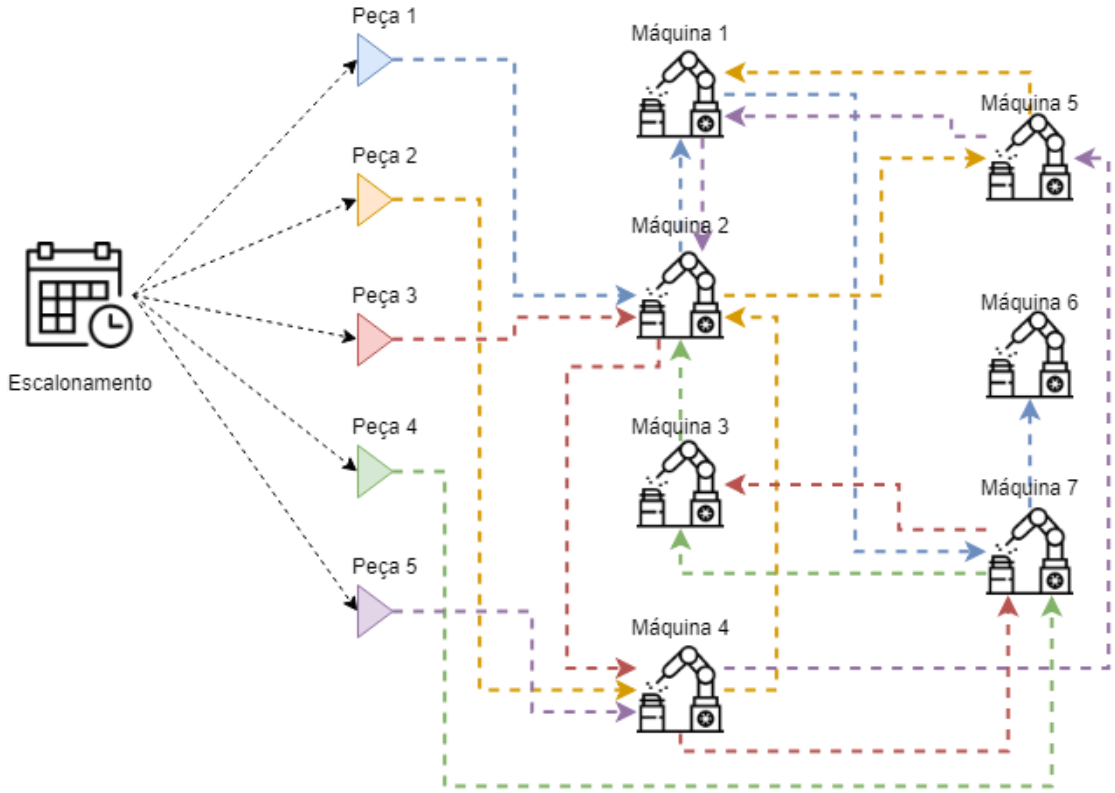

Figura 4.3 - Esquema geral do caso de estudo nº2

Tipo de peças

Foram consideradas 5 diferentes tipos de peça. Cada uma possui atributos distintos, como tempos de processamento ou cor. Todas as peças são processadas apenas por 4 das 7 máquinas. Na tabela 4.3 é apresentada a sequência que cada peça tem de respeitar até estar finalizada.

Tabela 4.3 – Sequenciamento para cada tipo de peça no caso de estudo nº2

	Sequenciamento			
Peça 1	M2	M1	M7	M6
Peça 2	M4	M2	M5	M1
Peça 3	M2	M4	M7	M3
Peça 4	M7	M3	M2	M4
Peça 5	M4	M5	M1	M2

Lançamento de peças

Tal como no caso de estudo nº1, o lançamento de peças é realizado no início do sistema de produção, sem que sejam considerados *stocks* de peças. O tempo de lançamento de peças para o sistema inicial é constante e igual a 1,5 minutos, isto é, a cada 1,5 minutos é lançada uma nova peça. Mais uma vez este tempo pode ser modificado para que sejam enviadas peças mais ou menos regularmente, tendo em conta se se pretende acumular mais ou menos peças nos *buffers* de entrada das máquinas.

Processamento

Como já referido o processamento das peças é efetuado em 4 máquinas. Cada máquina apenas processa uma peça de cada vez. Na tabela 4.4 são descritos os tempos de processamento iniciais de cada peça em cada máquina, representando apenas o tempo que estão efetivamente em processamento sem contabilizar tempos de espera nos *buffers*. Estes tempos foram determinados de forma aleatória, com valores entre 2 e 10 minutos. Também estes tempos serão modificados posteriormente para análise de diferentes cenários e testar o impacto da alteração dos tempos de produção na abordagem.

Tabela 4.4 - Tempos de processamento em minutos de cada peça para o caso de estudo nº2

	M1	M2	M3	M4	M5	M6	M7
Peça 1	9	2	-	-	-	8	7
Peça 2	5	9	-	10	5	-	-
Peça 3	-	6	7	10	-	-	7
Peça 4	-	7	7	5	-	-	3
Peça 5	9	2	-	10	6	-	-

Tal como no primeiro caso de estudo, antes e depois de processarem, as peças são inseridas em *buffers* segundo a lógica FIFO (F*irst-In-First-Out*). A capacidade dos *buffers* de saída é ilimitada e a capacidade de cada *buffer* de entrada de cada máquina é limitada, sendo inicialmente os seus valores os da tabela 4.4. Estes valores de capacidade também serão alterados para testes de diferentes cenários e medir o seu impacto nos resultados.

Tabela 4.5 - Capacidade do *buffer* de entrada em cada máquina para o caso de estudo nº2

	Capacidade
Máquina 1	5
Máquina 2	5
Máquina 3	5
Máquina 4	5
Máquina 5	5
Máquina 6	5
Máquina 7	5

Finalização

Neste caso de estudo, a finalização é o momento em que a peça sai da última máquina do seu sequenciamento. Mais uma vez, após serem finalizadas saem do sistema e são contabilizadas em armazém.

A Rede Neuronal deste caso de estudo, baseada na pesquisa bibliográfica do capítulo 2 e na abordagem metodológica do capítulo 3, apresenta uma topologia semelhante à anterior. Também são consideradas apenas duas camadas, entrada e saída.

A camada de entrada possui 35 neurónios. Cada neurónio é representativo do número de peças de cada tipo em cada máquina, como são 5 peças diferentes e 7 máquinas, dá-se um total de 35 neurónios a considerar.

A camada de saída possui 5 neurónios, representando o tipo de peça que deve ser escolhido para ser lançado. Também são considerados os pesos de *bias*, um por cada neurónio de saída, ficando a Rede Neuronal com um total de 180 pesos.

4.3 – Caso de estudo nº3

Este caso de estudo consiste num modelo de um caso de estudo real, tendo em conta um modelo de simulação adaptado de uma parte de um chão de fábrica de uma empresa de semicondutores. O modelo foi adaptado do modelo desenvolvido por um aluno de mestrado, no âmbito da sua dissertação, para otimização e simulação de sistemas de logística interna de um caso real [66].

Um modelo de um caso de estudo real assume um papel diferente dos casos anteriores, pois tem em conta mais caraterísticas e é esperado que o escalonamento seja mais complexo. Este modelo, representado de uma forma geral na figura 4.4, é constituído por áreas de trabalho, transportes por AGVs, lotes de peças e um armazém, do qual irão ser detalhadas as suas caraterísticas.

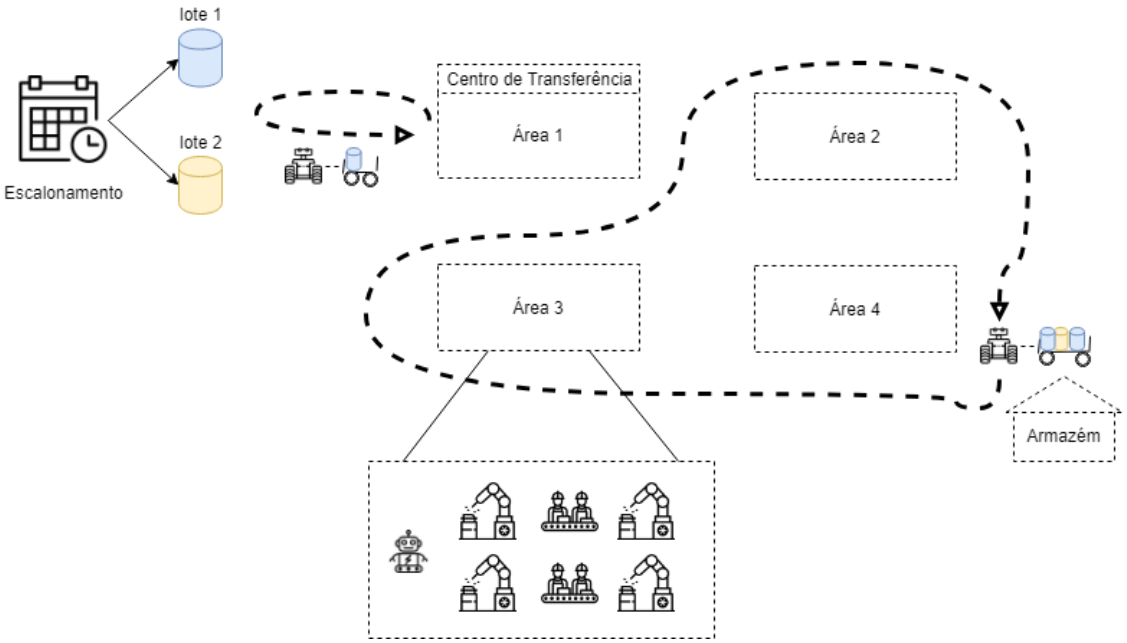

Figura 4.4 - Esquema geral do caso de estudo nº3

Lotes

Neste caso de estudo existem dois tipos de lotes. Um lote representa um conjunto de peças com as mesmas caraterísticas que são operadas em conjunto nas áreas. No modelo apenas é considerado o lote e não as peças que cada lote possui. Cada lote é processado sequencialmente em três áreas, conforme mostra a tabela 4.3.

Tabela 4.3 – Sequenciamento para cada tipo de lote

	Sequenciamento		
Lote 1	Área 2	Área 3	Área 4
Lote 2	Área 1	Área 2	Área 3

Escolha de lotes

A escolha do lote a ser lançado no sistema é realizada no início do sistema de produção, sem que sejam considerados *stocks*. O tempo de lançamento de um tipo de lote para o sistema inicial é constante e igual a 1,5 minutos, isto é, a cada 1,5 minutos é lançado um novo lote para ser transportado e processado nas diferentes áreas.

Transportes

Existem dois transportes a considerar no modelo. Tal como representado na figura 4.4, existem dois tipos de AGVs, cada um deles com uma função de transporte diferente. Um AGV para carregar lotes da produção até às áreas (AGV1) e outro que circula entre as áreas e armazém (AGV2).

É importante referir algumas caraterísticas importantes de cada tipo de AGV, como a capacidade de transporte, a velocidade de deslocamento, o número de AGVs de cada tipo e o instante em que é lançado o próximo AGV de cada tipo.

Tabela 4.4 - Caraterísticas de cada tipo de AGV

	Capacidade	Velocidade	População	Lançamento
AGV1	1 lote	4 m/s	1	0 minutos
AGV2	3 lotes	4 m/s	2	30 minutos

Sobre o AGV do tipo 2 é importante destacar que o seu funcionamento é similar ao de um metro, isto é, em cada área este recolhe os lotes que estão prontos para o próximo destino e deixa aqueles que o destino é compatível com a sequência.

Áreas

Cada área de produção representa uma zona do chão de fábrica em que as operações são semelhantes. Todas as áreas são compostas por operadores, máquinas e transportadores. Em cada área existe uma fila de entrada com capacidade de 5 lotes e outra de saída com capacidade infinita. Para cada área são considerados tempos de processamento para cada tipo de lote, presentes na tabela 4.5, que representam o tempo que cada tipo de lote passa dentro da área.

Tabela 4.5 - Tempos de processamento em minutos de cada lote em cada área

	Área 1	Área 2	Área 3	Área 4
Lote 1	-	1.5	2.5	2
Lote 2	2	1.5	2	-

Centro de Transferência

O centro de transferência funciona como *buffer* das áreas 2, 3 e 4. Quando um lote é transportado pelo AGV1, depois de ser selecionado para ser processado, se a sua sequência começar na área 1 este é diretamente transportado para lá, mas caso o destino seja uma das outras áreas é deixado no centro de transferência e tem de esperar que o AGV2 o recolha e deixe na respetiva área.

Finalização

No presente caso de estudo, a finalização é o momento em que o lote sai da última área do seu sequenciamento e é descarregada pelo AGV no armazém. No armazém são contabilizados quantos lotes totais foram produzidos.

A Rede Neuronal deste caso de estudo apresenta uma topologia semelhante às anteriores. Também são consideradas apenas duas camadas, entrada e saída.

A camada de entrada possui 10 neurónios. Cada neurónio é representativo do número de lotes de cada tipo em cada área ou centro de transferência. Visto existirem 2 tipos de lotes, 4 áreas e um centro de transferência, resultam 10 neurónios a serem considerados.

A camada de saída possui 2 neurónios, representando o tipo de lote que deve ser escolhido para ser transportado até às áreas pelo AGV. Também são considerados os pesos de *bias*, um por cada neurónio de saída, ficando a Rede Neuronal com um total de 12 pesos.

Capítulo 5

Análise de Resultados

Este capítulo expõe os resultados computacionais obtidos, com recurso a simulações de eventos discretos de vários cenários de teste, já apresentados no capítulo anterior. Os resultados surgem da execução dos cenários a partir de API, e posteriormente validação dos melhores resultados, substituindo os parâmetros necessários no Simio.

É pertinente referir que os cenários de teste usados para obtenção de resultados foram criados para que seja analisada a metodologia seguida para resolução de problemas de escalonamento de um chão de fábrica. Os resultados serão analisados por cada caso de estudo, sendo que em cada caso de estudo serão expostas as caraterísticas de cada cenário, bem como cada regra de prioridade usada para comparação. A regra de prioridade apenas cria uma solução e por isso o seu valor mantém-se constante ao longo das iterações.

Por fim, é importante referir que os resultados a serem apresentados foram resultado da execução, tanto do Simio versão 9.147 como do Visual Studio 2017, num computador ASUS K550J, com 12GB de memória RAM e um processador Intel Core i7-4710HQ (*quad-core* com velocidade até 3,5 GHz), em execução *single-threaded*.

5.1 – Caso de estudo nº1

Para o primeiro caso de estudo considerado no capítulo acima no ponto 4.1, é primeiramente feita uma análise dos cenários no Simio sendo complementada pela metodologia *Neuroevolution* (NE). Os diferentes cenários são apresentados por número e são descritas as caraterísticas de cada um.

Regra de prioridade

Para teste de eficiência do método foi desenvolvida uma regra de prioridade para fins de comparação. A regra consiste em enviar para produção um tipo de peça cujo destino é a máquina que contém menos peças em espera para processamento. Esta regra será testada em todos os cenários referente ao modelo nº1.

Cenário 1

Visto se tratar de um problema apenas com três máquinas, adequa-se a criação de um cenário simples com várias caraterísticas idênticas. Neste primeiro cenário é importante destacar algumas caraterísticas relevantes:

- Capacidade do *buffer* de entrada das máquinas de 5 peças;
- Capacidade de processamento em cada máquina de 1 peça;
- Tempos de processamento da tabela 4.1;
- Processamento de peças baseada em FIFO (*First In First Out*);
- Tempo de lançamento de peças de 1,5 minutos.

Após simulação do cenário descrito no Simio e posterior aplicação da abordagem foram extraídos os resultados. Ao fim de 5 iterações e tendo em conta uma população inicial de 10 Redes Neuronais os resultados obtidos a partir de simulações experimentais por API estão presentes na figura 5.1.

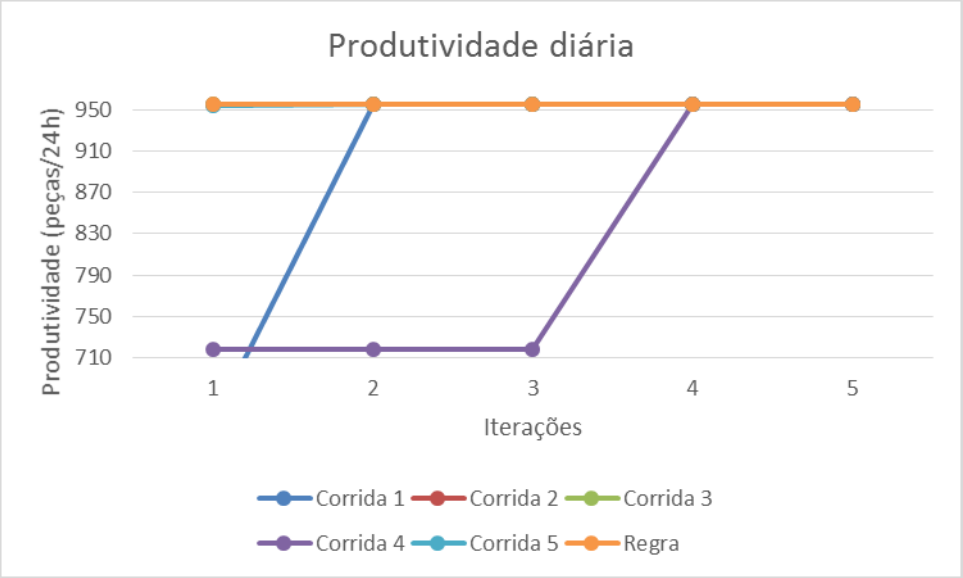

Figura 5.1 - Resultados de produtividade para o caso de estudo nº1

É possível verificar que foi obtido o mesmo valor final de produtividade após aplicação da regra de prioridade e para todas as corridas a partir de API da abordagem seguida de 956 peças produzidas em 24h de simulação. As corridas 1, 2 e 3 não estão visíveis pois obtiveram o mesmo valor que a regra de prioridade logo na primeira iteração, ficando assim sobrepostas.

Os resultados obtidos na execução deste primeiro cenário permitiram desde logo validar o correto funcionamento da abordagem desenvolvida. Por um lado, verifica-se que, ao longo das iterações, o valor de produtividade obtido pelas redes neuronais manteve-se ou cresceu, conforme esperado. Por outro lado, o desempenho final das redes atinge os mesmos valores de produtividade da regra desenvolvida especialmente para o caso de uso, podendo-se assim dizer que têm um bom desempenho.

Cenário 2

Para análise do impacto do tempo de processamento na aplicação da metodologia, foram mantidas as caraterísticas comuns do cenário 1 e alterados os tempos de processamento em cada máquina, presentes na tabela 5.3, para que haja uma diferença significativa entre eles.

Tabela 5.1 - Novos tempos de processamento em minutos para o caso de estudo nº1

	Máquina 1	Máquina 2	Máquina 3
Peça 1	10	-	-
Peça 2	-	1	-
Peça 3	-	-	10

Após serem alteradas as caraterísticas no modelo de simulação, ao fim de 5 iterações e tendo em conta uma população inicial de 10 Redes Neuronais, os resultados obtidos a partir de simulações experimentais por API estão presentes na figura 5.2.

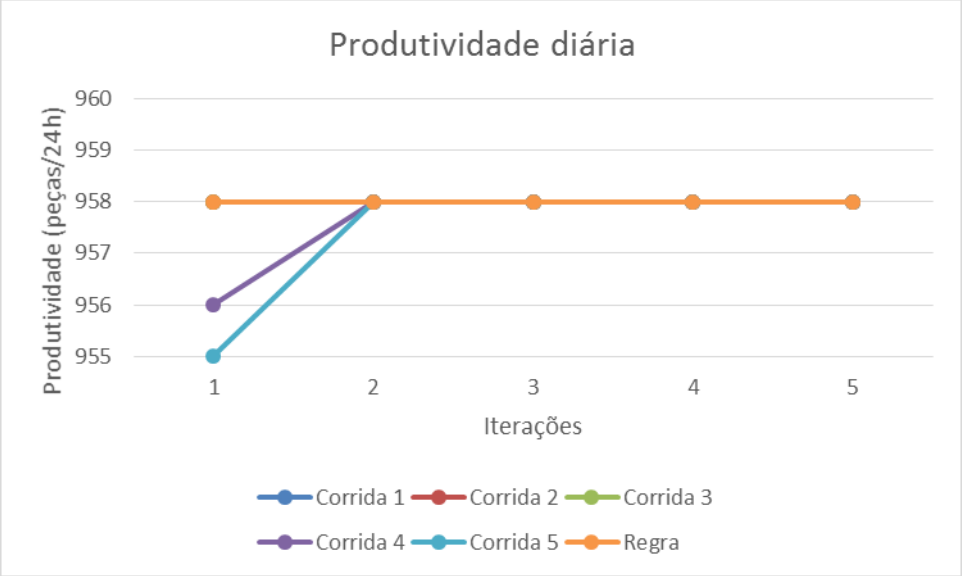

Figura 5.2 – Resultados da alteração dos tempos de processamento na produtividade para o caso de estudo nº1

Foi obtido o mesmo valor final de produtividade, tanto por aplicação da regra de prioridade como para todas as execuções da abordagem seguida, de 958 peças produzidas em 24h de simulação. Mais uma vez existiram corridas sobrepostas com a regra, nomeadamente as corridas 1, 2 e 3.

Cenário 3

Averiguado o impacto da mudança dos tempos de processamento na produtividade total, sugeriu-se que a capacidade dos *buffers* de entrada poderá também ter impacto na produtividade. Tendo presente os tempos de processamento da tabela 4.1 e as caraterísticas comuns aos cenários anteriores, são escolhidos aleatoriamente valores entre 1 e 10 peças para

as capacidades dos *buffers* e apresentados na tabela 5.4. Também o tempo de lançamento de peças foi reduzido para 1 minuto para que haja acumulação de peças em fila de espera.

Tabela 5.2 - Novos valores da capacidade dos *buffers* para o caso de estudo nº1

	Capacidade
Máquina 1	7
Máquina 2	3
Máquina 3	10

Prosseguindo com a alteração do cenário e aplicando o mesmo número de iterações, um dos resultados de produtividade alcançados a partir de simulações experimentais por API está presente na figura 5.3.

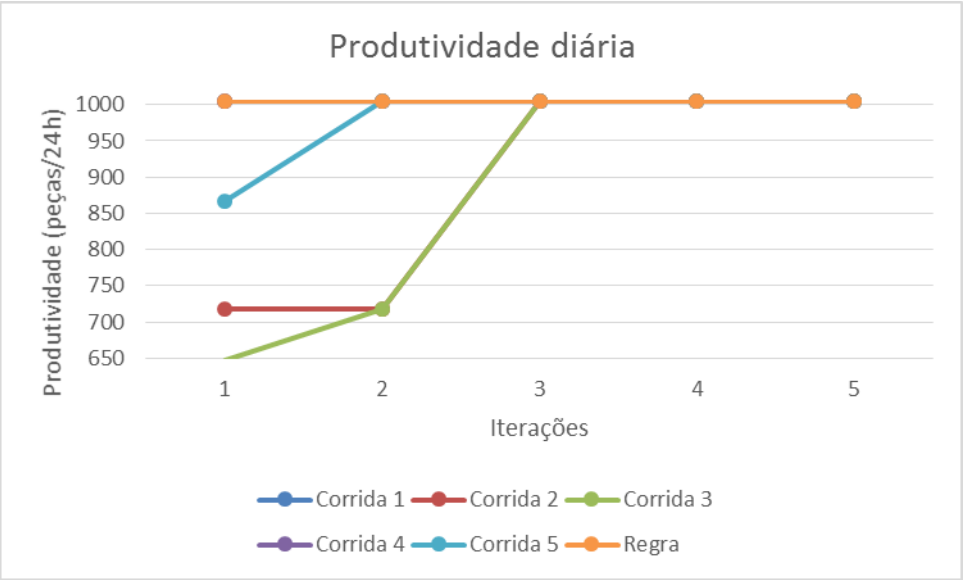

Figura 5.3 - Resultados da alteração da capacidade na produtividade para o caso de estudo nº1

Foi obtida uma produtividade de 1004 peças por parte da regra de prioridade e de 1005 por todas as execuções da abordagem seguida em 24h de simulação. Existe novamente sobreposição de corridas com a regra.

Análise global do caso de estudo nº1

Os resultados obtidos valida a utilização da abordagem, pois esta foi capaz de obter resultados com qualidade igual à das regras de prioridade criadas por humanos. A variação de caraterísticas do caso de estudo, como os tempos de processamento e a capacidade, não tiveram impacto no desempenho da abordagem desenvolvida, uma vez que foram na mesma obtidos os mesmos valores que a regra de prioridade.

Ao contrário do que se esperava nos cenários 2 e 3, a NE foi incapaz de obter melhor desempenho que a regra. A análise dos resultados permitiu concluir que, pelo facto dos *buffers* terem capacidade para acumular algumas peças, combinado com um tempo de lançamento menor que o tempo de produção médio, a regra criada era suficiente para obter a solução ótima para o problema (as máquinas estavam sempre a trabalhar).

Assim, pode-se afirmar que o caso de estudo tem um grau de complexidade tão baixo que não permite uma avaliação completa do potencial da abordagem NE, sendo assim necessário realizar testes em casos de estudo mais complicados.

5.2 – Caso de estudo nº2

Para o segundo caso de estudo é importante ter em conta as caraterísticas descritas no ponto 4.2 do capítulo 4. Serão consideradas 7 máquinas e 5 diferentes tipos de peças, as quais apenas são processadas em algumas máquinas e com sequenciamento diferente. Os diferentes cenários tal como o primeiro caso de estudo servem para análise de impacto da variação de alguns fatores no comportamento da abordagem.

Regra de prioridade

Tratando-se de um caso de estudo complexo, com várias máquinas e várias sequências para cada peça, foi elaborada uma regra de prioridade diferente da anteriormente definida. A regra consiste em verificar a cada instante de decisão a primeira máquina da sequência das peças e optar pela que possuir menor conteúdo no *buffer* de entrada. Caso a primeira máquina na sequência seja igual para dois tipos de peça, é verificada a máquina seguinte e por aí adiante. Também caso exista uma capacidade igual entre duas máquinas é escolhida qualquer um dos tipos de peça correspondentes à sequência. Esta regra será usada para comparação em todos os cenários deste caso de estudo.

Cenário 1

Este cenário foi criado com o intuito de compreender a dificuldade em atingir valores de produtividade satisfatórios aquando da implementação da abordagem em cenário mais complexos. Face às caraterísticas do caso de estudo, é importante destacar algumas caraterísticas relevantes:

- Todas as peças são processadas em 4 máquinas;
- Capacidade do *buffer* de entrada das máquinas de 5 peças;
- Capacidade de processamento em cada máquina de 1 peça;
- Tempos de processamento da tabela 4.4;
- Processamento de peças baseada em FIFO (*First In First Out*);
- Tempo de lançamento de peças de 1,5 minutos.

Derivado ao número elevado de pesos a considerar para construção de uma Rede Neuronal, explícitos na caraterização do caso de estudo, o número de iterações foi aumentado para 10 iterações. Como não é sabido qual o valor ideal de Redes Neuronais a serem consideradas na população inicial, serão feitos vários testes com um número de redes diferente e mantendo o número de iterações.

Começou-se por criar a população inicial com 10 Redes neuronais e ao fim de 5 corridas do algoritmo a partir de API, a figura 5.4 apresenta os resultados obtidos. Ao aplicar a regra de prioridade à simulação obteve-se um total de 235 peças em 24h de simulação.

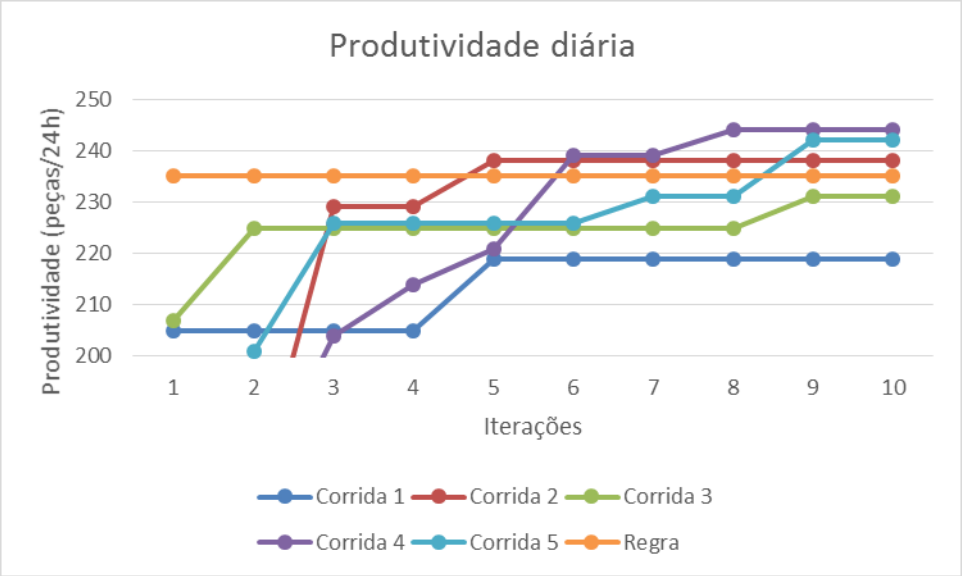

Figura 5.4 - Valores de produtividade para o caso de estudo nº2 com uma população inicial de 10 Redes Neuronais

Verificou-se que, para este cenário, a performance da NE foi melhor que a da regra desenvolvida, o que era um dos objetivos na criação deste caso mais complexo. Ainda que os resultados sejam desde já positivos, considerou-se necessário testar variações nos parâmetros da NE, a fim de melhorar o seu desempenho. Deste modo, depois de verificados os resultados anteriores, foram tidas em conta uma população inicial de 15 Redes Neuronais, mantendo o mesmo número de iterações. Os resultados obtidos ao fim de 5 corridas do algoritmo a partir de API estão presentes na figura 5.5.

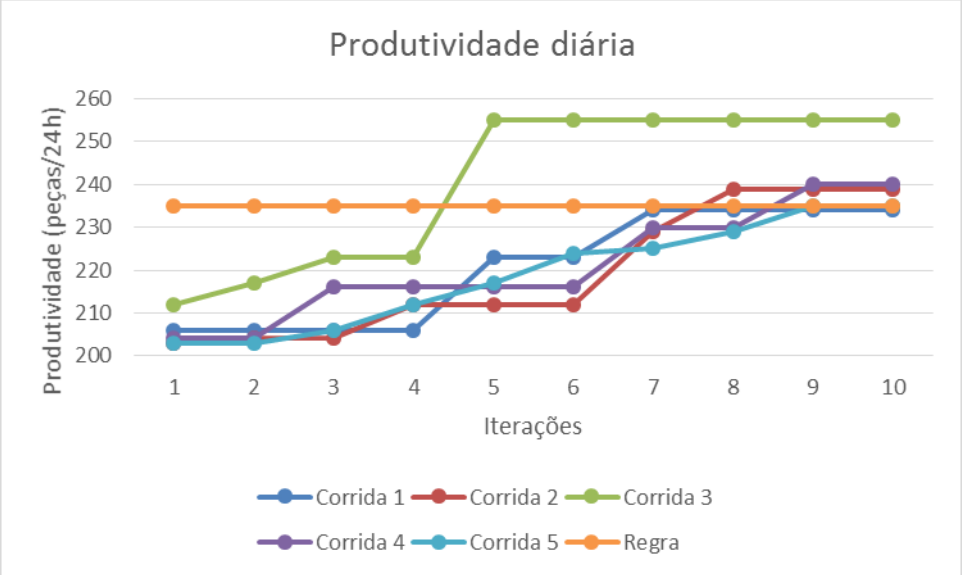

Figura 5.5 - Valores de produtividade para o caso de estudo nº2 com uma população inicial de 15 Redes Neuronais

Aumentando mais uma vez o número de redes neuronais de 15 para 20, para analisar o seu impacto, foram obtidos os resultados da figura 5.6.

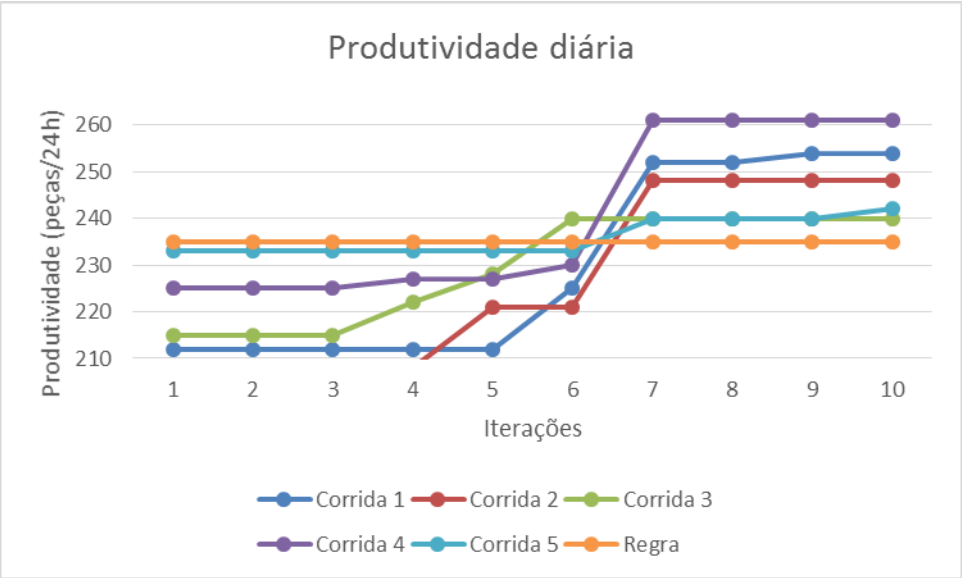

Figura 5.6 - Valores de produtividade para o caso de estudo nº2 com uma população inicial de 20 Redes Neuronais

Por fim, mais uma vez foi aumentado o número de Redes neuronais de 20 para 25 e os resultados de produtividade, ao fim de 24h de simulação de produção, estão presentes na figura 5.7.

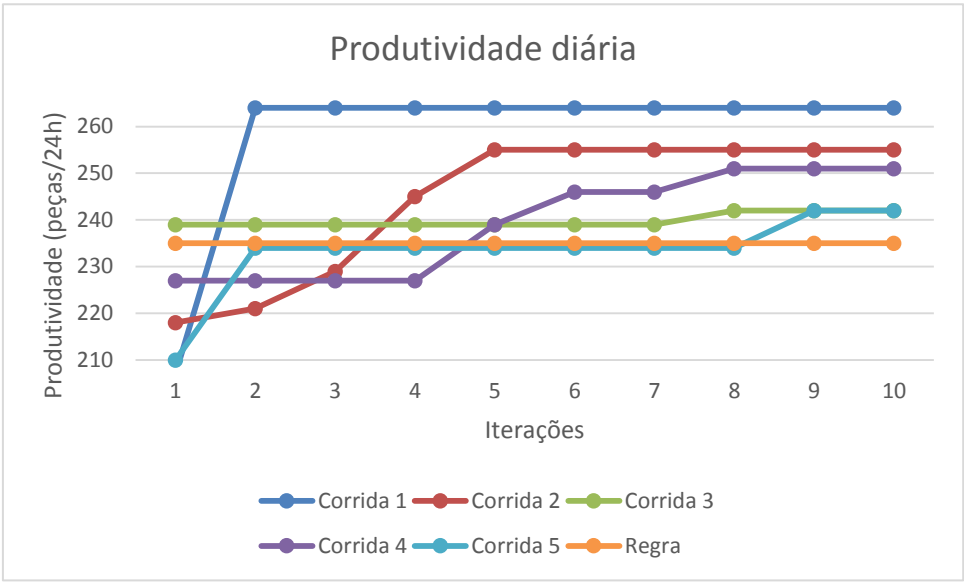

Figura 5.7- Valores de produtividade para o caso de estudo nº2 com uma população inicial de 25 Redes Neuronais

No fim deste cenário é pertinente ter em conta todos os valores finais de produtividade alcançados com a variação do número de redes, visível na tabela 5.3. Com base nos resultados é percetível que o número de redes afeta o desempenho do algoritmo, sendo que foram obtidos melhores resultados à medida que se aumentou o número de redes da população inicial.

Tabela 5.3- Valores finais de produtividade de cada corrida para cada valor da população de Redes Neuronais e tempo total decorrido em minutos

	10 Redes	15 Redes	20 Redes	25 Redes
Corrida 1	219	234	254	264
Corrida 2	238	239	248	255
Corrida 3	231	255	240	242
Corrida 4	244	240	261	251
Corrida 5	242	235	242	242
Valor médio	234,8	240,6	249,0	250,8
Execução temporal	17 min	25 min	36 min	48 min

É importante salientar que o aumento do número de Redes Neuronais também implicou um esforço computacional crescente, tornando o processo mais lento. Por outro lado, analisando os valores médios da tabela 5.3, é percetível que existiu um aumento da eficácia, fazendo com que exista claramente um *trade-off* entre eficiência e eficácia.

Com base apenas no valor médio das produtividades, é oportuno considerar que a escolha de 25 Redes Neuronais demonstra-se ser um valor viável, quando em comparação com os outros valores. No entanto o esforço computacional, que leva a que o processo demore mais tempo, induz que também são viáveis as soluções com maior equilíbrio entre eficiência e eficácia, optando-se assim por soluções de 15 ou 20 Redes Neuronais.

Cenário 2

Com base em alguns dos resultados verificados no cenário 1 deste caso de estudo, em que foi possível averiguar que o número de redes da população inicial poderia influenciar os resultados obtidos, considerou-se interessante proceder à análise do número de iterações para possível determinação de um critério de paragem do método. Para tal, todas as outras caraterísticas do caso de estudo nº2 foram mais uma vez consideradas.

Para este cenário foram usadas para a população inicial 15 Redes Neuronais, por apresentarem um equilíbrio interessante entre eficácia e o período de tempo em que são obtidos os resultados. Como apenas é alterado o número de iterações, a regra de prioridade mantém o valor de 235 peças em 24h.

Uma vez que se pretende uma análise mais aprofundada ao número de iterações, foram consideradas mais corridas do algoritmo em API. Com um total de 10 corridas, foram extraídos os resultados das mesmas e demonstrados em duas figuras (figura 5.8 e 5.9) apenas para que não exista uma sobreposição maior de corridas e seja mais fácil a sua análise.

Por fim, para auxiliar a análise ao caso de estudo, será apresentada uma tabela com os valores médios de produtividade deste cenário, bem como a taxa de aumento ao longo das iterações.

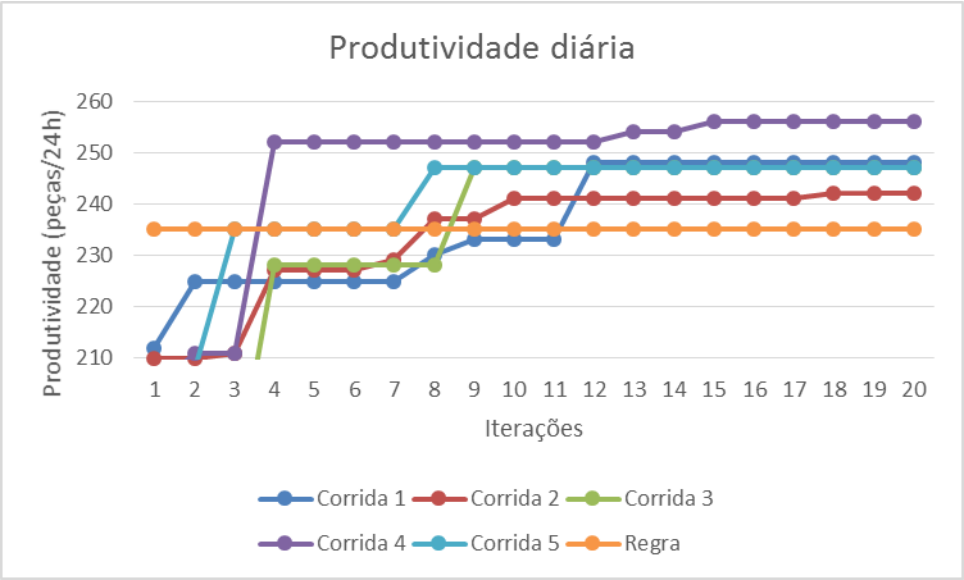

Figura 5.8 - Valores de produtividade das 5 primeiras corridas para o cenário nº2 ao fim de 20 iterações

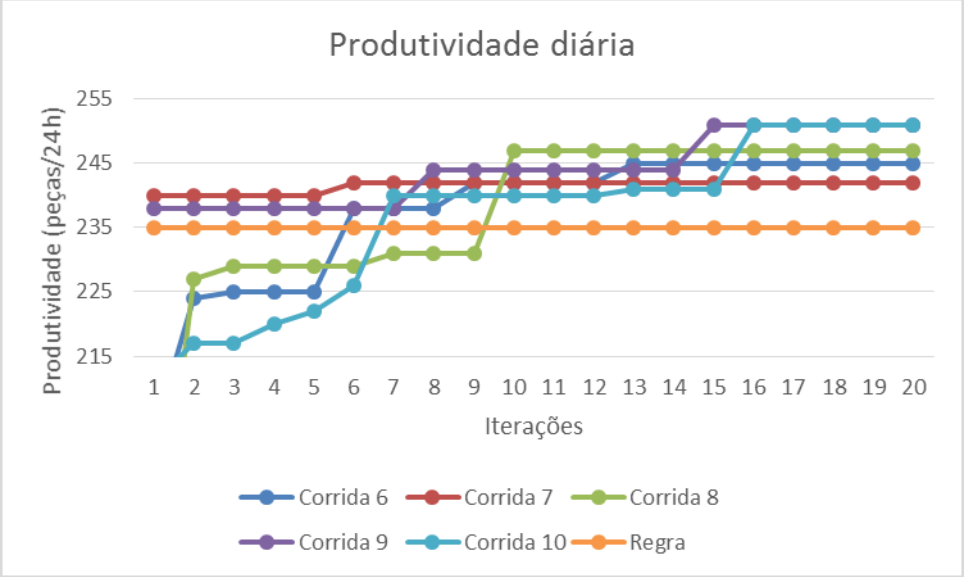

Figura 5.9 - Valores de produtividade das 5 últimas corridas para o cenário nº2 ao fim de 20 iterações

Na tabela 5.4 são apresentados os valores médios obtidos das 10 corridas a cada iteração. O aumento é calculado tendo em conta o valor da iteração correspondente com o valor médio da iteração anterior. O valor é mostrado em percentagem e não existe aumento para a primeira iteração por não existir iteração precedente.

Tabela 5.4 – Impacto do número de iterações no caso de estudo nº2

Iteração	Valor médio	Aumento
1	206,8	-
2	218,5	5,35%
3	221,6	1,40%
4	231,9	4,44%
5	232,1	0,09%
6	234,0	0,81%
7	235,8	0,76%
8	238,9	1,30%
9	241,5	1,08%
10	243,5	0,82%
11	243,5	0,00%
12	245,0	0,61%
13	245,6	0,24%
14	245,6	0,00%
15	246,5	0,37%
16	247,5	0,40%
17	247,5	0,00%
18	247,6	0,04%
19	247,6	0,00%
20	247,6	0,00%

Por análise da tabela é visível que as últimas iterações do método não conseguem aumentar a produtividade, considerada em valor médio. Posto isto, o valor ideal de iterações para que se obtenham boas soluções e se despenda o menor tempo é de cerca de 16 iterações.

Análise global do caso de estudo nº2

Este caso de estudo permitiu tirar uma série de conclusões além do que foi observável no primeiro caso.

Primeiramente, o maior grau de complexidade deste caso de estudo permitiu o desenvolvimento de uma abordagem de NE capaz de um desempenho melhor que o de uma regra criada especificamente para o problema, validando mais profundamente a utilização desta mesma abordagem para este tipo de problemas.

Além disso, com este caso de estudo, foi ainda possível fazer um estudo do impacto de alguns parâmetros da NE na sua performance, tendo-se conseguido analisar o *trade-off* entre a qualidade da solução e o esforço computacional e chegar a valores mais adequados para o número de redes neuronais a utilizar e de iterações do programa.

5.3 – Caso de estudo nº3

Para este caso de estudo foram consideradas as caraterísticas descritas no ponto 4.3 do capítulo 4, em que o modelo descrito é constituído por áreas de trabalho, transportes por AGVs,

lotes de peças e um armazém. Tratando-se de um modelo de um caso de estudo real o acréscimo de tempo de simulação é inevitável. Face aos tempos de processamento, capacidades e sequenciamento apresentados, é aplicada a metodologia *Neuroevolution* para que se avalie a sua implementação em casos reais.

Regra de prioridade

Também para este caso de estudo foi elaborada uma regra de prioridade que determina o tipo de lote a ser lançado consoante as filas de espera estão ou não ocupadas. Tratando-se de um caso de estudo de um modelo complexo, a regra consiste em comparar as zonas onde se juntam os lotes, isto é, como a sequência de cada lote é diferente mas existe partilha das áreas 2 e 3, é importante decidir o tipo de lote a lançar para que exista o menor acumular possível. Primeiro é comparado o número de lotes da primeira área de cada sequência e enviado o lote correspondente à área com menores conteúdos. Depois é comparado o número de peças da área 2 com os da área 3, uma vez que a área 3 é a última da sequência do lote 2, se existirem mais lotes na área 3 que na área 2 é enviado tipo 2, caso contrário é enviado tipo 1.

Para além dessas caraterísticas também é importante considerar que o tempo de simulação foi reduzido, de 24 horas para 12 horas, e de 12 horas para 6 horas, devido ao elevado tempo que demora a executar cada uma das simulações. Primeiramente foi considerada uma população inicial de 10 Redes Neuronais e foram realizadas 5 iterações. Para 12 horas e sendo executadas 5 corridas, foram obtidos os valores presentes na figura 5.10. Para os valores da figura 5.11 foram consideradas 6 horas de simulação, mas foram aumentadas para 10 o número de iterações e para 15 o número de Redes Neuronais.

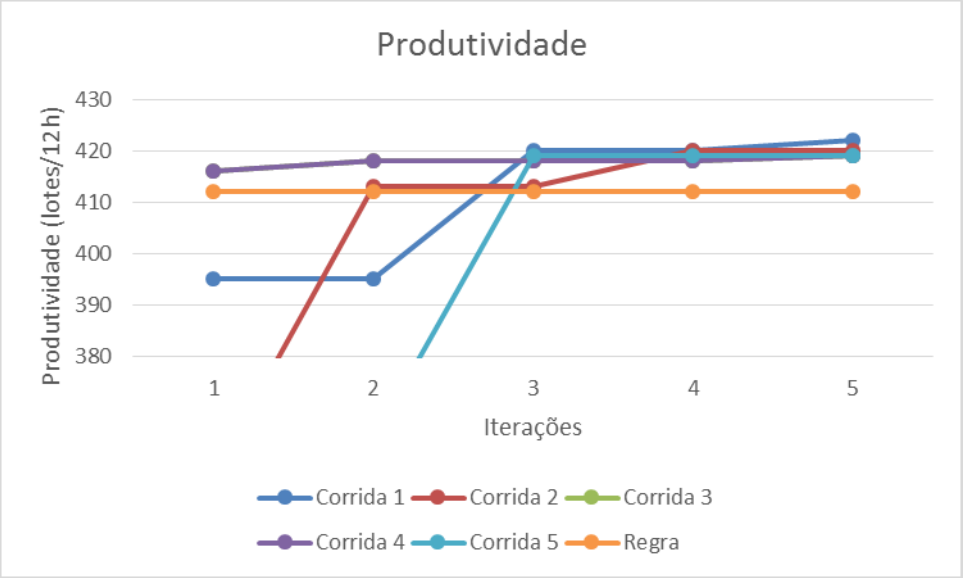

Figura 5.10 – Valores de produtividade em 12 horas de simulação para o caso de estudo nº3

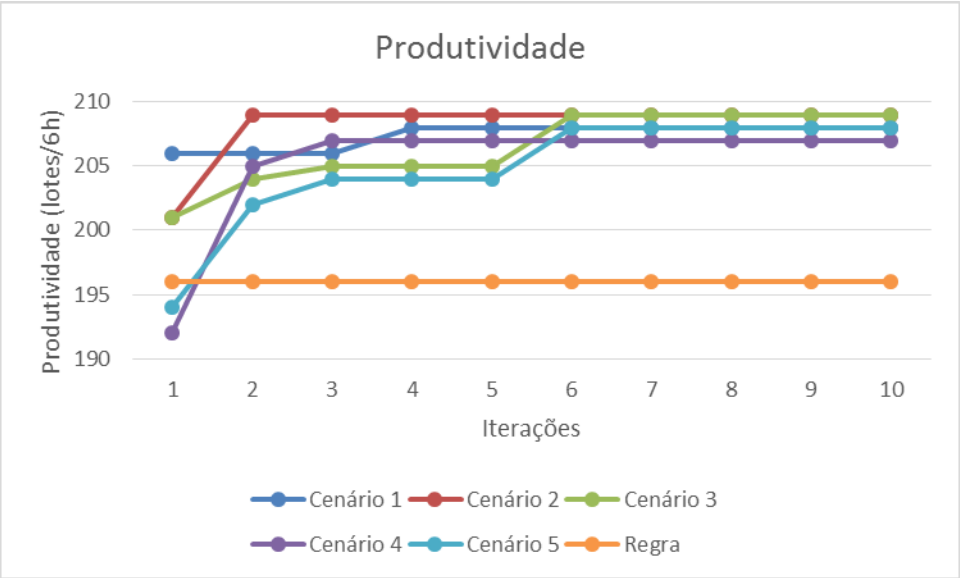

Figura 5.11 - Valores de produtividade em 6 horas de simulação para o caso de estudo nº3

Após extraídos os resultados, tendo em conta diferentes iterações e a complexidade de um problema real foram obtidos resultados finais superiores aos obtidos por uma regra de prioridade. No primeiro caso a regra de prioridade atingiu um valor de 212 lotes produzidos e o melhor caso da aplicação abordagem ao fim de 5 iterações foi de 222 lotes. Para o segundo caso a regra obteve uma produtividade de 196 lotes e o melhor caso da aplicação da abordagem atingiu 209 lotes.

Apesar dos bons resultados apresentados, a complexidade da modelação de um caso de estudo real leva a que haja limitações a nível temporal. Para o primeiro caso, onde são apenas testadas 5 iterações de 12 horas de tempo simulado, foram necessárias 2 horas e 20 minutos para que se obtivesse o resultado das 5 corridas.

Tratando-se de um caso de estudo referente real, ainda que adaptado e sem incorporar todas as características do problema existente, este último caso permitiu fazer um primeiro teste à aplicabilidade desta metodologia para problemas atuais de chão de fábrica. Os resultados foram promissores, tendo-se obtido um desempenho melhor do que aquele gerado pela regra de prioridade.

Capítulo 6

Conclusão e Trabalho Futuro

Neste capítulo são apresentadas as conclusões referentes ao trabalho realizado ao longo do projeto. Serão analisadas as contribuições desta dissertação e avaliados se os objetivos propostos foram cumpridos. Para além de conclusões, também são dadas algumas sugestões para trabalho futuro, tendo presente uma mentalidade assente na melhoria contínua e com o objetivo de compreender o que ainda pode ser feito.

6.1 – Conclusões

O principal objetivo desta dissertação passava pela elaboração de um método inovador, baseado em *Machine Learning* e simulação, para resolução de problemas de escalonamento de um chão de fábrica. O desempenho do método seria medido em comparação com regras de prioridade simples.

A abordagem seguida baseou-se em usar Algoritmos Genéticos para evoluir Redes Neuronais, técnica de *Reinforcement Learning* conhecida como *Neuroevolution*. Primeiramente foi construída uma topologia para as Redes Neuronais a desenvolver e de seguida adaptada cada rede aos diferentes casos de estudo. O recurso a algoritmos genéticos permitiu ajustar os pesos de cada rede construída para que os resultados caminhassem no sentido do objetivo pretendido.

Integrando esta metodologia com simulação de eventos discretos no Simio, foram obtidos resultados satisfatórios. É importante referir também que a análise dos resultados possibilitou retirar conclusões acerca da eficácia da metodologia. A metodologia usada resolveu com sucesso todos os casos de estudo propostos, incluindo um caso de estudo real.

No primeiro caso de estudo, caraterizado por três máquinas e três peças com apenas processamento em uma delas por cada peça, obtiveram-se resultados que igualaram a regra de prioridade, usada para comparação, e não existiu uma melhoria significativa devido à simplicidade do caso.

No caso seguinte, onde a complexidade em desenvolver um escalonamento a um chão de fábrica aumentou, foram comparados vários cenários para validar a adequabilidade da abordagem. Obtidos os resultados, concluiu-se não só que a NE poderia ser usada para este tipo de problemas, como também se retiraram conclusões interessantes sobre o número de Redes Neuronais a considerar e ao ponto de paragem no número de iterações em que o algoritmo estabiliza. O número de Redes Neuronais a considerar depende muito dos recursos

computacionais e temporais disponíveis, devido ao facto de que o aumento do número de redes traduz-se no aumento da eficácia, mas implica o aumento do tempo despendido também. Quanto às iterações, perante o caso apresentado, chegou-se à conclusão que em média são necessárias 16 iterações para que o algoritmo convirja.

Por fim, aplicando a abordagem a um caso de estudo real, a metodologia apresentou mais uma vez resultados satisfatórios, tendo melhor desempenho que a regra criada, o que reflete a eficácia do método como sendo uma mais valia a aplicar em casos futuros.

Uma vez atingidos os objetivos propostos, o desempenho desta abordagem apresenta um bom indicador da possível viabilidade do uso desta metodologia por parte do INESC TEC no futuro, para projetos que envolvam problemas complexos de otimização numa realidade de chão de fábrica.

6.2 – Trabalho futuro

Cada vez mais a aposta em métodos inovadores colhe frutos ao nível da produção, obtendo resultados que ainda não tinham sido alcançados. É com esta mentalidade que o trabalho futuro deve ser pensado.

Um aspeto que pode ser visto como problema na metodologia usada nesta dissertação está relacionado com o tempo despendido pela simulação em modelos mais complexos, como é o caso de um modelo real. Assim, e no que toca à parte da simulação, uma das medidas interessantes a implementar consiste em encontrar uma forma de acelerar o processo de simulação ou mesmo testar outro tipo de programas para simulação de eventos discretos, quando se tratam de modelos com muita informação, tal como Anylogic.

Na vertente da inteligência artificial, uma medida interessante a aplicar no futuro consiste na implementação de outro tipo de técnicas de *Reinforcement Learning* para comparação com a metodologia da presente dissertação, tal como *Q-Learning*. Além disso, é importante continuar o trabalho de validação da abordagem, aplicando-a a outros problemas complexos e, preferencialmente, que digam respeito a casos de estudo reais.

Finalmente, no que toca ao algoritmo de NE propriamente dito, pequenas alterações, como escolha de novos operadores de seleção, *crossover* e mutação ou arquiteturas de rede com mais camadas intermédias, poderão ser testadas e levar a uma melhoria de desempenho.

Referências

[1] M. Gen and L. Lin, "Multiobjective evolutionary algorithm for manufacturing scheduling problems: State-of-the-art survey," *J. Intell. Manuf.*, vol. 25, no. 5, pp. 849–866, 2014.

[2] M. L. Pinedo, *Scheduling: Theory, algorithms, and systems*. 2008.

[3] K. R. Baker and D. Trietsch, *Principles of Sequencing and Scheduling*. 2009.

[4] T. Wuest, D. Weimer, C. Irgens, and K.-D. Thoben, "Machine learning in manufacturing: advantages, challenges, and applications," *Prod. Manuf. Res.*, vol. 4, no. 1, pp. 23–45, 2016.

[5] P. Brucker, *Scheduling Algorithms*, vol. 93. 2007.

[6] Y. Mati, S. Dauzère-Pérès, and C. Lahlou, "A general approach for optimizing regular criteria in the job-shop scheduling problem," *Eur. J. Oper. Res.*, vol. 212, no. 1, pp. 33–42, 2011.

[7] T. C. Chiang and L. C. Fu, "Using dispatching rules for job shop scheduling with due-date based objectives," *Int. J. Prod. Res.*, vol. 45, pp. 3245–3262, 2007.

[8] A. Arisha, P. Young, and M. El Baradie, "Job Shop Scheduling Problem : an Overview," in *International Conference for Flexible Automation and Intelligent Manufacturing (FAIM 01)*, 2001, pp. 682–693.

[9] P. Priore, A. Gómez, R. Pino, and R. Rosillo, "Dynamic scheduling of manufacturing systems using machine learning: An updated review," *Artif. Intell. Eng. Des. Anal. Manuf.*, vol. 28, no. 01, pp. 83–97, 2014.

[10] D. C. Mattfeld, "Evolutionary search and the job shop : investigations on genetic algorithms for production scheduling," 1996.

[11] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker, "Complexity of Machine Scheduling Problems," *Annals of Discrete Mathematics*, vol. 1, no. C. pp. 343–362, 1977.

[12] J. Grabowski, E. Nowicki, and S. Zdrzałka, "A block approach for single-machine scheduling with release dates and due dates," *Eur. J. Oper. Res.*, vol. 26, no. 2, pp. 278–285, 1986.

[13] H. M. Wagner, "An Integer Linear Programming Model for Machine Scheduling," *Nav. Res. Logist. Q.*, vol. 6, no. 2, pp. 131–140, 1959.

[14] É. D. Taillard, "Parallel Taboo Search Techniques for the Job Shop Scheduling Problem," *ORSA J. Comput.*, vol. 6, no. 2, pp. 108–117, 1994.

[15] J. Adams, E. Balas, and D. Zawack, "The Shifting Bottleneck Procedure for Job Shop Scheduling," *Manage. Sci.*, vol. 34, no. 3, pp. 391–401, 1988.

[16] J. Carlier and E. Pinson, "An Algorithm for Solving the Job-Shop Problem," *Manage. Sci.*, vol. 35, no. 2, pp. 164–176, 1989.

[17] D. J. Hoitomt, P. B. Luh, and K. R. Pattipati, "A Practical Approach to Job-Shop Scheduling Problems," *IEEE Trans. Robot. Autom.*, vol. 9, no. 1, pp. 1–13, 1993.

[18] S. S. Panwalkar and W. Iskander, "A Survey of Scheduling Rules," *Oper. Res.*, vol. 25, no. 1, pp. 45–61, 1977.

[19] K. E. Stecke and J. J. Solberg, "Loading and control policies for a flexible manufacturing system," *Int. J. Prod. Res.*, vol. 19, no. 5, pp. 481–490, 1981.

[20] J. H. Blackstone, D. T. Phillips, and G. L. Hogg, "A state-of-the-art survey of dispatching rules for manufacturing job shop operations," *Int. J. Prod. Res.*, vol. 20,

no. 1, pp. 27–45, 1982.
[21] K. R. Baker, “Sequencing Rules and Due-Date Assignments in a Job Shop,” *Manage. Sci.*, vol. 30, no. 9, pp. 1093–1104, 1984.
[22] T. E. M. Ari P. J. Vepsalainen, “Priority Rules for Job Shops with Weighted Tardiness Costs,” *Manage. Sci.*, vol. 33, no. 8, pp. 1035–1047, 1987.
[23] L. L. Tang, Y. W. Yih, and C. Y. Liu, “A Study on Decision Rules of a Scheduling Model in an Fms,” *Comput. Ind.*, vol. 22, no. 1, pp. 1–13, 1993.
[24] H. Cho and R. A. Wysk, “A robust adaptive scheduler for an intelligent workstation controller,” *Int. J. Prod. Res.*, vol. 31, no. 4, pp. 771–789, 1993.
[25] F. Meziane, S. Vadera, K. Kobbacy, and N. Proudlove, “Intelligent systems in manufacturing: current developments and future prospects,” *Integr. Manuf. Syst.*, vol. 11, no. 4, pp. 218–238, 2000.
[26] H. C. Zhang and S. H. Huang, “Applications of neural networks in manufacturing: A state-of-the-art survey,” *Int. J. Prod. Res.*, vol. 33, no. 3, pp. 705–728, 1995.
[27] S. Webster, P. D. Jog, and A. Gupta, “A genetic algorithm for scheduling job families on a single machine with arbitrary earliness/tardiness penalties and an unrestricted common due date,” *Int. J. Prod. Res.*, vol. 36, no. 9, pp. 2543–2551, 1998.
[28] M. A. B. Candido, “A genetic algorithm based procedure for more realistic job shop scheduling problems,” *Int. J. Prod. Res.*, vol. 36, no. 12, pp. 3437–3457, 1998.
[29] L. Min and W. Cheng, “A genetic algorithm for minimizing the makespan in the case of scheduling identical parallel machines,” *Artif. Intell. Eng.*, vol. 13, no. 4, pp. 399–403, 1999.
[30] L. Yu, H. M. Shih, and T. Sekiguchi, “Fuzzy inference-based multiple criteria FMS scheduling,” *Int. J. Prod. Res.*, vol. 37, no. 10, pp. 2315–2333, 1999.
[31] P. Cunningham and B. Smyth, “Case-based reasoning in scheduling: Reusing solution components,” *Int. J. Prod. Res.*, vol. 35, no. 11, pp. 2947–2962, 1997.
[32] T. Gabel and M. Riedmiller, “Scaling adaptive agent-based reactive job-shop scheduling to large-scale problems,” in *Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling, CI-Sched 2007*, 2007, pp. 259–266.
[33] Y. C. Wang and J. M. Usher, “Learning policies for single machine job dispatching,” *Robot. Comput. Integr. Manuf.*, vol. 20, no. 6 SPEC. ISS., pp. 553–562, 2004.
[34] E. Szelke and G. Márkus, “A learning reactive scheduler using CBR/L,” *Comput. Ind.*, vol. 33, pp. 31–46, 1997.
[35] C. O. Kim, H. S. Min, and Y. Yih, “Integration of inductive learning and neural networks for multi-objective FMS scheduling,” *Int. J. Prod. Res.*, vol. 36, no. 9, pp. 2497–2509, 1998.
[36] C.-Y. Lee, S. Piramuthu, and Y.-K. Tsai, “Job shop scheduling with a genetic algorithm and machine learning,” *Int. J. Prod. Res.*, vol. 35, no. 4, pp. 1171–1191, 1997.
[37] Y. Li, “Deep Reinforcement Learning: An Overview,” pp. 1–70, 2017.
[38] R. S. Sutton and a G. Barto, “Reinforcement learning: an introduction.,” *IEEE Trans. Neural Netw.*, vol. 9, no. 5, p. 1054, 1998.
[39] L. Busnoniu, R. Babuska, B. De Schutter, and D. Ernst, Reinforcement learning and dynamic programming using function approximators. 2010.
[40] T. Gabel and M. Riedmiller, “Adaptive Reactive Job-Shop Scheduling with Learning Agents,” *Int. J. Inf. Technol. Intell. Comput.*, vol. 2, no. 4, 2007.
[41] D. E. Moriarty, A. C. Schultz, and J. J. Grefenstette, “Evolutionary Algorithms for Reinforcement Learning,” *J. Artif. Intell. Res.*, vol. 11, pp. 241–276, 1999.
[42] R. Miikkulainen, “Neuroevolution,” *Encyclopedia of Machine Learning*. pp. 1–6, 2010.
[43] G. F. Luger, *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*, vol. 5th. 2005.
[44] D. Floreano, P. Dürr, and C. Mattiussi, “Neuroevolution: From architectures to learning,” *Evolutionary Intelligence*, vol. 1, no. 1. pp. 47–62, 2008.
[45] J. Mao, *Why artificial neural networks?* 1996.
[46] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, vol. Addison-We. 1989.
[47] J. Carr, “An Introduction to Genetic Algorithms,” *Whitman Coll. Math. Dep.*, pp. 1–40, 2014.
[48] O. Kramer, *Genetic Algorithm Essentials*, vol. 679. 2017.

[49] K. O. Stanley and R. Miikkulainen, “Evolving Neural Networks through Augmenting Topologies,” *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, 2002.
[50] K. Anjaria, “Neuroevolution For Effective Information Security Training,” in *International Conference on Soft Computing and its Engineering Applications*, 2017.
[51] M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone, “A Neuroevolution Approach to General Atari Game Playing,” *IEEE Trans. Comput. Intell. AI Games*, vol. 6, no. 4, pp. 1–18, 2013.
[52] K. O. Stanley, R. Cornelius, R. Miikkulainen, T. D. Silva, and A. Gold, “Real-time Learning in the NERO Video Game,” in *Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference*, 2005, vol. 2003, no. 6, pp. 2003–2004.
[53] L. Cardamone, D. Loiacono, and P. L. Lanzi, “Learning to drive in the open racing car simulator using online neuroevolution,” *IEEE Trans. Comput. Intell. AI Games*, vol. 2, no. 3, pp. 176–190, 2010.
[54] C. J. C. H. Watkins and P. Dayan, “Q-learning,” *Mach. Learn.*, vol. 8, no. 3–4, pp. 279–292, 1992.
[55] Z. Rubo, G. U. Guochang, L. I. U. Zhaode, and W. Xingce, “Reinforcement Learning Theory , Algorithms and Its Application,” 2000, vol. 8152, no. 1, pp. 1143–1146.
[56] Y. Zhan, H. B. Ammar, and M. E. Taylor, “Theoretically-grounded policy advice from multiple teachers in reinforcement learning settings with applications to negative transfer,” in *IJCAI International Joint Conference on Artificial Intelligence*, 2016, vol. 2016–Janua, no. 7540, pp. 2315–2321.
[57] V. Mnih *et al.*, “Playing Atari with Deep Reinforcement Learning Volodymyr,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[58] S. C. Wang, Z. X. Song, H. Ding, and H. Bin Shi, “An improved reinforcement Q-learning method with BP neural networks in robot soccer,” *Proc. - 2011 4th Int. Symp. Comput. Intell. Des. Isc. 2011*, vol. 1, no. 4, pp. 177–180, 2011.
[59] H. Zhu, I. Ch Paschalidis, and M. E. Hasselmo, “Feature Extraction in Q-Learning using Neural Networks *,” in *IEEE 56th Annual Conference on Decision and Control (CDC)*, 2017, no. Cdc, pp. 3330–3335.
[60] F. Petroski, S. Vashisht, M. Edoardo, C. Joel, L. Kenneth, and O. S. Jeff, “Deep Neuroevolution : Genetic Algorithms are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning,” 2017.
[61] J. Andersson, “Environmental Activity Based Cost Using Discret Event Simulation,” in *Proceedings of the 2011 Winter Simulation Conference*, 2011, pp. 891–902.
[62] J. Banks, “Discrete Event Simulation,” in *Proceedings of the 1999 Winter Simulation Conference*, 1999, pp. 7–13.
[63] M. Semini and J. O. Strandhagen, “Applications Of Discrete-Event Simulation To Support Manufacturing Logistics Decision-Making: A Survey,” in *Proceedings of the 2006 Winter Simulation Conference*, 2006, pp. 1946–1953.
[64] J. A. (North C. S. U. Joines and S. D. (North C. S. U. Roberts, *Simulation Modeling with SIMIO: A Workbook*. 2013.
[65] Stackoverflow, “How to evolve weights of a neural network in neuroevolution.” [Online]. Disponível em : https://stackoverflow.com/questions/31708478/how-to-evolve-weights-of-a-neural-network-in-neuroevolution.
[66] T. Santos, “Otimização e simulação de sistemas de logística interna – caso real de definição de rotas milk run numa empresa de semicondutores,” Faculdade de Engenharia Da Universidade Do Porto, 2018.